System-Level Modeling and Synthesis Techniques for Flow-Based Microfluidic Very Large Scale Integration Biochips

Minhass, Wajid Hassan

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

System-Level Modeling and Synthesis Techniques for Flow-Based Microfluidic Very Large Scale Integration Biochips

Wajid Hassan Minhass

Kongens Lyngby 2012
IMM-PHD-2012-286

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

To My Parents

Summary

Microfluidic biochips integrate different biochemical analysis functionalities on-chip and offer several advantages over the conventional biochemical laboratories. In this thesis, we focus on the flow-based biochips. The basic building block of such a chip is a valve which can be fabricated at very high densities, e.g., 1 million valves per cm^2. By combining these valves, more complex units such as mixers, switches, multiplexers can be built up and the technology is therefore referred to as microfluidic Very Large Scale Integration (mVLSI).

The manufacturing technology for the mVLSI biochips has advanced faster than Moore's law. However, the design methodologies are still manual and bottom-up. Designers use drawing tools, e.g., AutoCAD, to manually design the chip. In order to run the experiments, applications are manually mapped onto the valves of the chips (analogous to exposure of gate-level details in electronic integrated circuits). Since mVLSI chips can easily have thousands of valves, the manual process can be very time-consuming, error-prone and result in inefficient designs and mappings.

We propose, for the first time to our knowledge, a top-down modeling and synthesis methodology for the mVLSI biochips. We propose a modeling framework for the components and the biochip architecture. Using these models, we present an architectural synthesis methodology (covering steps from the schematic design to the physical synthesis), generating an application-specific mVLSI biochip. We also propose a framework for mapping the biochemical applications onto the mVLSI biochips, binding and scheduling the operations and performing fluid routing. A control synthesis framework for determining the exact valve activation sequence required to execute the application is also proposed. In order to reduce the macro-assembly around the chip and enhance chip scalability, we propose an approach for the biochip pin count minimization. We also propose a throughput maximization scheme for the cell culture mVLSI biochips, saving time and reducing costs. We have extensively evaluated the

proposed approaches using real-life case studies and synthetic benchmarks. The proposed framework is expected to facilitate programmability and automation, enabling the emergence of a large biochip market.

Resumé

Mikrofluidiske biochips integrerer forskellige funktionaliteter til biokemiske analyser på en chip og har adskillige fordele sammenlignet med konventionelle biokemiske laboratorier. I denne afhandling fokuserer vi på flow-baserede biochips. Den grundlæggende byggesten i sådan en chip er ventilen (størrelse: $6{\times}6\ \mu m^2$), der kan fremstilles med en densitet på 1 million ventiler pr cm^2. Ved at kombinere disse ventiler kan man fremstille komplekse enheder som mixere, skiftere og multipleksere og teknologien bliver derfor kaldt mikrofluidisk Very Large Scale Integration (mVLSI).

Fremstillingsteknologien for mVLSI biochips har udviklet sig hurtigere end Moore's lov. Dog er designmetodologierne stadig manuelle og bottom-up. Designere bruger tegneværktøjer, fx. AutoCAD, til manuelt at designe chippen. For at køre eksperimenter bliver applikationen manuelt mapped på chippens ventiler (sammenligneligt med gate-level design i elektroniske ICs). Da mVLSI chips nemt kan have tusindvis af ventiler kan denne manuelle process være meget tidskrævende, tilbøjelig til fejl og resultere i ineffektive designs og mappings.

Vi foreslår, for første gang så vidt vi ved, en top-down modellerings og syntese metodologi til mVLSI biochips. Vi foreslår et modellerings framework for komponenterne og biochiparkitekturen. Ved brug af disse modeller præsenterer vi en arkitektural syntesemetodologi (som dækker trinene fra det skematiske design til den fysiske syntese), hvilket resulterer i en applikationsspecifik mVLSI biochip. Vi foreslår også et framework til at mappe den biokemiske applikation på mVLSI biochips, hvor vi binder og planlægger operationerne og router fluiderne. Et kontrolsynteseframework til at afgøre den præcise aktiveringssekvens af ventilerne for at afvikle applikationen bliver også foreslået. For at reducere macro-assembly omkring chippen og forbedre skalerbarheden foreslår vi en metode til at minimisere pin count. Vi har evalueret de foreslåede metoder i stor udstrækning ved brug af real-life case studies og syntetiske benchmarks. Det foreslåede framework forventes at lette programmerbarhed og automation.

Preface

This thesis was prepared at the Department of Informatics and Mathematical Modelling, the Technical University of Denmark in partial fulfilment of the requirements for acquiring the Ph.D. degree in engineering.

The thesis proposes a top-down synthesis methodology for the modeling and synthesis of the flow-based microfluidic Very Large Scale Integration (mVLSI) biochips. The work has been supervised by Associate Professor Paul Pop and co-supervised by Professor Jan Madsen.

Kongens Lyngby, November 2012

Wajid Hassan Minhass

Papers included in the thesis

- Wajid Hassan Minhass, Paul Pop and Jan Madsen. System-Level Modeling and Synthesis of Flow-Based Microfluidic Biochips. *Proceedings of the Compilers, Architecture, and Synthesis for Embedded Systems Conference (CASES)*, pp. 225–234, 2011. Published.

- Wajid Hassan Minhass, Paul Pop, Jan Madsen, Mette Hemmingsen, Peder Skafte-Pedersen and Martin Dufva. Cell Culture Microfluidic Biochips: Experimental Throughput Maximization. *Proceedings of the IEEE International Conference on Bioinformatics and Biomedical Engineering (iCBBE)*, pp. 1–6, 2011. Published.

- Wajid Hassan Minhass, Paul Pop, Jan Madsen and Felician Stefan Blaga. Architectural Synthesis of Flow-Based Microfluidic Large-Scale Integration Biochips. *Proceedings of the Compilers, Architecture, and Synthesis for Embedded Systems Conference (CASES)*, pp. 181–190, 2012. Published.

- Wajid Hassan Minhass, Paul Pop and Jan Madsen. Synthesis of Biochemical Applications on Flow-Based Microfluidic Biochips using Constraint Programming. *Proceedings of the IEEE Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*, pp. 37–41, 2012. Published.

- Wajid Hassan Minhass, Paul Pop, Jan Madsen and Tsung-Yi Ho. Control Synthesis for the Flow-Based Microfluidic Large-Scale Integration Biochips. *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013. Published.

- Wajid Hassan Minhass, Paul Pop and Jan Madsen.System-Level Modeling and Application Mapping for Flow-Based Microfluidic Very Large Scale Integration Biochips. In preparation for submission to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD).*
- Wajid Hassan Minhass, Paul Pop and Jan Madsen. Application-Specific Architectural Synthesis for Flow-Based Microfluidic Very Large Scale Integration Biochips. In preparation for submission to *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*
- Wajid Hassan Minhass, Paul Pop, Jan Madsen and Tsung-Yi Ho. Control Synthesis and Pin Count Minimization for Flow-Based Microfluidic Very Large Scale Integration Biochips. In preparation for submission to *Journal on Emerging Technologies in Computing Systems (JETC).*

Acknowledgements

I would like to start off by sending out a resounding thanks to my supervisors Paul Pop and Jan Madsen for their exceptional guidance, invaluable close involvement with the work and seemingly boundless patience. Their ability to always find time to discuss ideas and give feedback, and to do so in the most amicable manner, contributed dearly in the completion of the thesis work.

I would also like to thank my colleagues at ESE for the pleasant and creative work environment. Special thanks to Elena Maftei, discussions with whom helped me immensely in developing a much deeper understanding of the work.

Last but not least, I wish to extend my grand and deepest gratitude to my parents, to whom this thesis is dedicated, my lovely wife and my siblings for their love and continuous support, which served as a strong driving force.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Microfluidics is the science of handling and manipulating very small volumes of fluids that are in the sub-millimeter scale. It is a multidisciplinary field that involves engineering, biotechnology, micro-technology and several others. Over the last 10 years, more than 35,000 papers have been published on the topic of microfluidics and the annual publication count is continuously on the rise [11] (see Figure 1.1). According to the ISI Web of Science, these papers currently receive over 65,000 citations per year (statistics for the citations in year 2011). In addition, over 1,500 patents referring to microfluidics have been issued only in USA [17]. It is evident that, in recent years, microfluidics has become a rapidly emerging and engaging topic for both academia and industry.

1.1 Microfluidic Biochips

Microfluidic biochips integrate different biochemical analysis functionalities (e.g., dispensers, filters, mixers, separators, detectors) on-chip, miniaturizing the macroscopic chemical and biological processes to a sub-millimetre scale [82]. These microsystems offer several advantages over the conventional biochemical analyzers, e.g., reduced sample and reagent volumes, speeded up biochemical reactions, ultra-sensitive detection and higher system throughput, with several assays being integrated on the same chip [87].

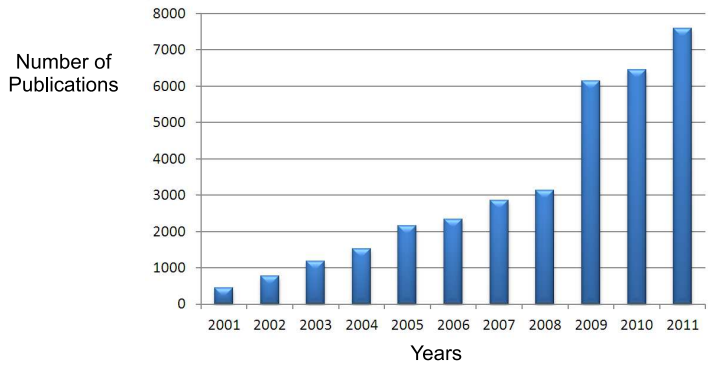


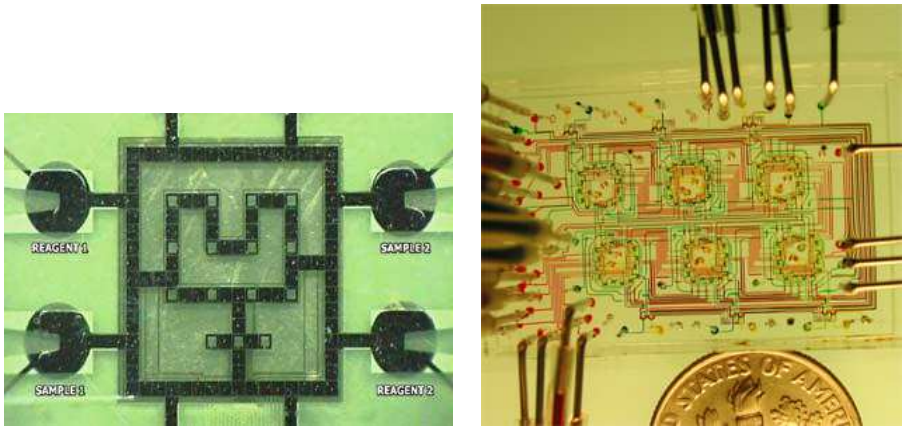
Figure 1.1: Publication Count Related to Microfluidics [11]

The roots of microfluidic technology go as far back as 1950s when the first efforts were made to dispense nanolitre and picolitre volumes of liquids [54]. This later formed the basis of today's ink-jet technology [46]. The year 1979 set a milestone in terms of fluid propulsion within microchannels of sub-millimeter cross-section by realizing a miniaturized gas chromatograph [80] on a silicon wafer. By the early 1990s, several microfluidic structures, e.g., micropumps, microvalves, had been realized by silicon micro-machining [50, 72]. This laid the foundation for automating biochemical protocols by integrating microfluidic structures and resulted in the advent of "micro total analysis systems" (μTAS) [52], also called "lab-on-a-chip" [36] or simply "microfluidic biochips".

There are several types of microfluidic biochip platforms, each having its own advantages and limitations [54]. Based on how the liquid is manipulated on the chip, biochips can be broadly divided into two types:

- Droplet-based biochips,
- Flow-based biochips.

In droplet-based biochips (also referred to as digital biochips) the liquid is manipulated as discrete droplets on a two dimensional array of electrodes [24, 79], see Figure 1.2a. Several techniques have been proposed for on-chip droplet manipulation [31]. Electrowetting-on-dielectric (EWOD) [67] is one of the most commonly used techniques. In EWOD, voltages are applied on the electrodes in a predefined way and the droplet movement is achieved by creating/ collapsing the electric field. The principle can be used to dispense droplets onto the chip and route them on the electrodes [67]. Adjacent set of electrodes can be combined together to form a virtual component, e.g., a mixer can be created by



(a) Droplet-based Biochip [79] (b) Flow-based Biochip [16]

Figure 1.2: Microfluidic Biochips

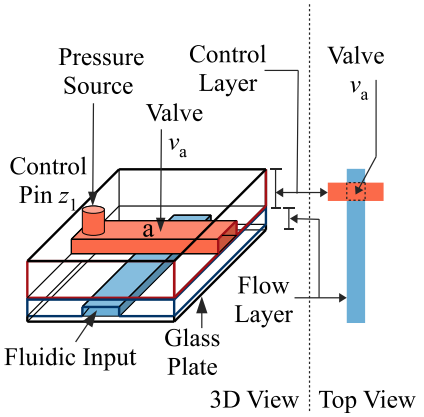
grouping adjacent electrodes and moving the droplet around on these electrodes in order to achieve mixing. Any set of electrodes can be used for this purpose and therefore the chip is termed reconfigurable [24]. The same electrodes can later be used for performing other operations as well, e.g., fluid transport, storage. More components, e.g., detectors (photo-diodes), can be added to the chip in order to achieve the desired functionality.

In this thesis, the focus is on the flow-based biochips, see Figure 1.2b. The following subsections explain the flow-based technology and its application areas.

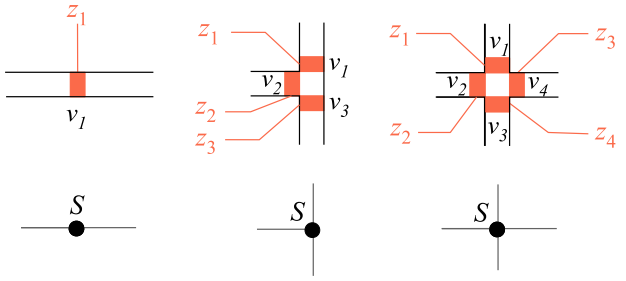
1.2 Flow-based mVLSI Biochips

Flow-based biochips are manufactured using multilayer soft lithography. A cheap, rubber-like elastomer (polydimethylsiloxane, PDMS) with good biocompatibility and optical transparency is used as the fabrication substrate [55]. Physically, the biochip can have multiple layers, but the layers are logically divided into two types: *flow layer* (depicted in blue in Figure 1.3a) and the *control layer* (depicted in red). The liquid in the flow layer is manipulated using the control layer [55].

The basic building block of such a biochip is a valve (see Figure 1.3a), which is used to manipulate the fluid in the flow layer as the valves restrict/ permit the fluid flow. The control layer (red) is connected to an external air pressure



(a) Microfluidic Valve



(b) Switch Configurations

Figure 1.3: Flow-Based Valve and Switch

source through the punch hole (called a control pin) z_1. The flow layer (blue) is connected to a fluid reservoir through a pump that generates the fluid flow. When the pressure source is not active, the fluid is permitted to flow freely (open valve). When the pressure source is activated, high pressure causes the elastic control layer to pinch the underlying flow layer (point a in Figure 1.3a) blocking the fluid flow (closed valve). A very thin membrane ($<1\mu m^2$) is sandwiched between the flow layer and the control layer providing flat geometry and hence more stabilization to the valve. Because of their small size ($6\times6\ \mu m^2$), these valves can be fabricated at densities approaching 1 million valves per cm^2 [23]. By combining several microvalves, more complex units such as switches, mixers, micropumps, multiplexers, etc., can be built up, with hundreds of units being accommodated on one single chip [54]. The technology is therefore referred to as "microfluidic Very Large Scale Integration" (mVLSI) [23].

One example of the valves combining to form a component is that of a switch (a mixer formed by combining valves is shown in Section 2.1.1). As shown in Figure 1.3b, a switch may consist of one valve (restricting/ allowing flow in a channel) or may consist of more than one valve. Multiple valve switches are present at the channel junctions and are used to control the path of the fluids entering the switch from different sides. The fluid flow can be generated using off-chip or on-chip pumps. The control layer can be placed both above and/ or below the flow layer, creating "push-down" or "push-up" valves, respectively. Connections to the external ports (fluidic ports and pressure sources) are made by punching holes in the chip (gaining access to the flow and control layer by creating flow pins and control pins) and placing external tubings (connected to the external fluidic reservoirs through pumps or pressure sources) into the punch holes [55]. All input ports are connected to the off-chip pumps.

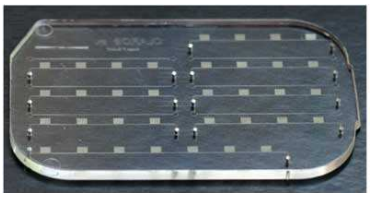
The mVLSI technology allows integrating multiple varying complexity components together in a seamless fashion (similar to digital electronics) in order to create a highly complex design, without requiring the knowledge of detailed properties of the manipulated liquids [55]. Using hundreds of thousands of microvalves, the chip provides exquisite control over its biological contents.

1.2.1 Application Areas

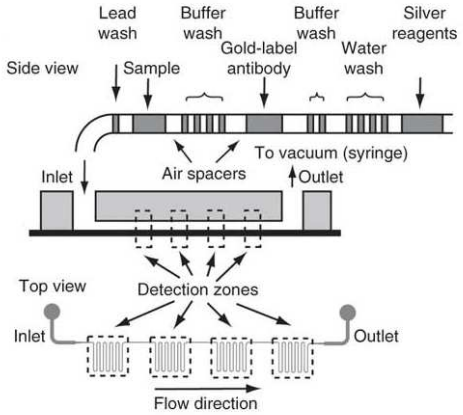
Since the advent of this technology, the designed chips have been used for a variety of applications [35, 38, 40, 47, 53]. Some of these are discussed below:

- *Drug Discovery*: These chips allow massively-parallel, high throughput testing of molecules, which is ideal for drug discovery. For example, in order for the hepatitis C virus to proliferate, one of its proteins needs to interact and bind with the RNA. The flow-based chip in [30] has been used to screen over 1,200 small molecules to test if such a protein-RNA interaction was inhibited and 14 such molecules were found. The results were later used to develop a drug which is now in clinical trials. The same strategy can be used for drug discovery for other diseases.

- *Diagnostic Testing*: The biochip shown in Figure 1.4 has been designed for testing HIV and syphilis [29]. The chip is cheap, easy to use, requires only micro-litres of the blood sample and it simultaneously tests for HIV and syphilis giving out the result within 20 minutes. The chip has been utilized successfully in Rwanda to test hundreds of locally collected human samples. In this chip, microfluidic procedures of fluid handling and signal detection (test does not require user interpretation of the signal) have been



(a) HIV biochip



(b) Reagent delivery on the chip

Figure 1.4: Flow-Based Biochip for HIV Detection [29]

integrated into a single, easy to use, point-of-care device that replicates all the steps of the current state-of-the-art, at a lower material cost [29].

- *Prenatal Screening*: In the chip in [32], proof of concept studies for a chip that can be used for non-invasive prenatal test (to test for chromosomal abnormalities) has been reported. The mother's blood is used to measure the fetal DNA. This chip has been used to successfully identify cases of trisomy 21 (Down syndrome), trisomy 18 (Edward syndrome) and trisomy 13 (Patau syndrome) [32]. A company, Verinata Health [18], was launched earlier this year to make this technology available to general public. Many other similar examples exist.

Microfluidic biochips can readily facilitate clinical diagnostics, especially immediate point-of-care disease diagnosis. In addition, they also offer exciting application opportunities in the realm of preventive individualized health-care, massively parallel DNA analysis, enzymatic and proteomic analysis, cancer and stem cell research, and automated drug discovery [33, 82]. Utilizing these biochips to perform food control testing, environmental (e.g., air and water samples) monitoring and biological weapons detection are also interesting possibilities.

Medical industry is one of the primary beneficiaries of the advancements in microfluidic biochips. The International Technology Roadmap for Semiconductors 2011 has listed "Medical" as a "Market Driver" for the future [10]. Many companies related to biochips have already emerged in recent years and have

reported significant profits. Major market players these days are Fluidigm Corporation [7], Affymetrix Inc. [1], Agilent Technologies [2], Caliper Life Sciences Inc. [5], GE Healthcare Ltd. [8], Illumina, Inc. [9], Life Technologies Corporation [13], among many others. According to the research report "Biochips: A Global Strategic Business Report" released by Global Industry Analysts, Inc. in March 2012, the global biochip market is expected to reach US $4.6 billion by 2017 [4].

Next section presents the motivation behind our research. Section 1.4 summarizes the thesis objectives and briefly outlines its contributions. An overview of the thesis is presented at the end of this chapter.

1.3 Motivation

Currently, designers are using full-custom and bottom-up methodologies involving many manual steps to implement these chips and to run the experiments. The following subsection gives a brief overview of the current practices.

1.3.1 Chip Design, Fabrication and Application Mapping

The mVLSI biochips are currently being designed manually using the drawing Computer-Aided Design (CAD) programs, e.g., AutoCAD [3]. The biochip foundries (Stanford [15], CalTech [12]) provide AutoCAD template files and a set of design rules in order to initiate the design process. The designer does the design manually by drawing lines representing microfluidic channels and circles representing punch holes in the chip (for accessing flow and control layers). The designer needs to have a complete understanding of the application in order to design a chip that fulfils the requirements. At the same time, he also needs to have the knowledge and skills of the chip design as it is his responsibility to ensure that all design rules, e.g., channel thickness and height, height to width aspect ratio, spacing between channels and punch holes, are satisfied as he manually does the placement and routing on the chip. Doing the design in this way is extremely time consuming and error-prone. Once the desired microfluidic chip has been designed, the AutoCAD file is then sent to the foundry for fabrication.

At the foundry, two separate molds (flow mold and control mold) are made using conventional photo-lithography techniques. Next, in order to fabricate the chip

(consider that the chip has push-down valves like the one shown in Figure 1.3a), following steps are followed:

- *Make Control Layer*: PDMS is poured onto the control mold in order to form a thick layer and it is then baked.
- *Make Flow Layer*: PDMS spinning is done onto the flow mold in order to form a thin layer and it is then baked.
- *Align Layers*: Control layer PDMS is peeled off from the control mold and is aligned on top of the flow layer on the flow mold. As shown in Figure 1.3a, the control layer is aligned on top of the flow layer in order ensure that the valve is formed at the right location. Note that a valve is formed only at the intersection of the flow channel and the control channel, provided that the area of intersection is large enough. Control channels of smaller width can easily pass over the flow channels without forming valves. It is the designer's task to ensure that the control channel is of the correct width in case a valve is desired.
- *Bond Layers*: The two layers are bonded together by further baking.
- *Bond Device to a Flat Plate*: The flow layer (that has the control layer on top of it) is now peeled off from the flow mold and bonded and sealed to a flat plate, typically a glass slide. Figure 1.3a shows the glass plate at the bottom of the flow layer.

The punch holes are made in the chip using a special punching device (punch hole locations are selected by the chip designer) in order to provide access to the flow and control layers. If it provides access to the flow layer, it is called a *flow pin*, and can be used as a fluid input or output port. The flow pins are connected to off-chip fluid reservoirs. If the punch hole provides access to the control layer, then it is called a *control pin*.

The control pins are connected to the off-chip pressure sources in order to control the opening and closing of valves for executing the desired application on the chip. Prof. Quake's group at Stanford University [15] has developed a USB-based valve control system to drive the valves in the chip from a computer using LabView or Matlab. This means that the user has to manually map the application onto the valves of the biochip (analogous to exposing the gate-level details in electronic ICs) [81]. A chip can easily have thousands of valves (the number of valves is rapidly increasing), therefore, the manual process is clearly very tedious and for larger chips and applications, the process can easily result in inefficient mappings. Also, the process needs to be repeated every time a change is made either to the chip architecture or the biochemical application.

As the chips grow more complex (commercial biochips are available which use more than 25,000 valves and about a million features to run 9,216 polymerase chain reactions in parallel [66]) and the need of having multiple and concurrent assays on the chip becomes more significant, these manual, bottom-up methodologies become highly inadequate. Therefore, new top-down design methodologies and design tools are needed in order to successfully manage the increase in design complexity. The electronic VLSI circuits have benefited heavily from design automation and the electronic designers today work as conveniently on the billion transistor multi-core processors as they did on the Intel 4004 processor with only 2,250 transistors in the early days. The CAD support for mVLSI biochips is expected to enable the emergence of a large biochip market.

1.3.2 Related Work

In academia, significant amount of work has been carried out on the individual microfluidic components [49, 54]. The manufacturing technology, soft lithography, used for the flow-based biochips has advanced faster than Moore's law [39]. Although biochips are becoming more complex everyday, Computer-Aided Design (CAD) tools for these chips are still in their infancy. Most CAD research has been focussed on device-level physical modeling of components [43, 73].

Significant work on top-down synthesis methodologies for droplet-based biochips has been proposed [24, 68]. However, the architecture of the droplet-based chips differs significantly from the flow-based chips. In the flow-based biochips, components of different types (e.g., mixers, heaters) are physically designed on the chip and connected to each other using microfluidic channels. Once fabricated, the number and type of the components, their placement scheme on the chip and the routing interconnections cannot be modified [55]. Droplet-based biochips (as discussed in Section 1.1), however, use the idea of virtual components and are reconfigurable. Because of the architectural differences, the models and techniques proposed for the digital biochips are not applicable to their flow-based counterparts.

The industry has gotten around the limited CAD tools problem by limiting the number of chips that they design and using them for multiple applications (Fluidigm Corporation has only 4 chip designs [6]). The soft lithography based fabrication process is, however, cheap and has a fast turn around time [55], pointing to having application-specific chips capable of providing higher efficiency instead of doing a multi-purpose design.

Figure 1.5 shows the typical electronic VLSI and mVLSI design cycles. Given the system specifications (e.g., application requirements, chip area), the mVLSI

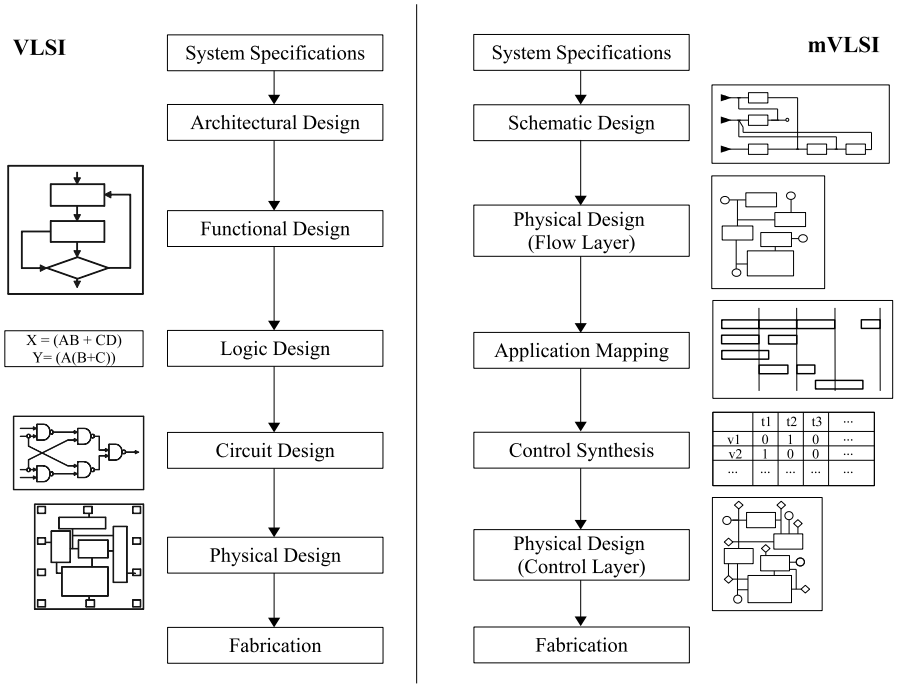


Figure 1.5: VLSI vs mVLSI Design Cycles

design starts by designing the schematic of the required biochip. This is followed by the physical synthesis of the flow layer, i.e., placement of components and routing of flow channels while following the design rules. After the flow channels have been routed, the channel lengths and therefore the routing latencies for the fluids that traverse these channels can now be calculated. Next, the given biochemical application is mapped onto this biochip architecture and the optimized schedule for its execution is generated. Based on the schedule, the control information (which valves to open and close at what time and for how long) can now be extracted. Optimization schemes can be used to share control pins between valves reducing the macro-assembly around the chip. Since the number of control pins and their sharing between valves is now known, the control layer can now be routed. Once the routing is complete, the chip design is ready to be sent for fabrication.

The details of the related work, with respect to each of the design tasks described above, are discussed in the respective chapters.

1.4 Thesis Objectives and Contributions

In order to obtain a scalable, top-down approach for the design of mVLSI biochips, the foremost step is to devise *models* for the biochip components, the biochip architecture as well as the biochemical applications that need to be executed on the chip. These models should be such that the design problems of synthesizing the biochip architectures, mapping of the biochemical applications to the mVLSI biochips and synthesizing the control for automatically executing the application on the designed biochip can be easily formulated. These models and the design problem formulations are the primary contribution of this thesis.

Figure 1.6 shows our proposed design methodology. Our contributions are briefly outlined below:

- *Modeling and Simulation*

  We propose a dual-layer modeling framework for the mVLSI components. The model captures the component operations at the flow layer as well as the control layer valve activations that are needed in order to execute these operations [59, 58] (the box "Component Library" in Figure 1.6). We propose a topology graph-based model for the mVLSI biochips that captures the chip components, their interconnections, the fluid flow paths on the chip and also the routing constraints. These models are used in all phases of the design methodology in Figure 1.6. For the biochemical applications we use a sequencing graph model (similar to the one used in

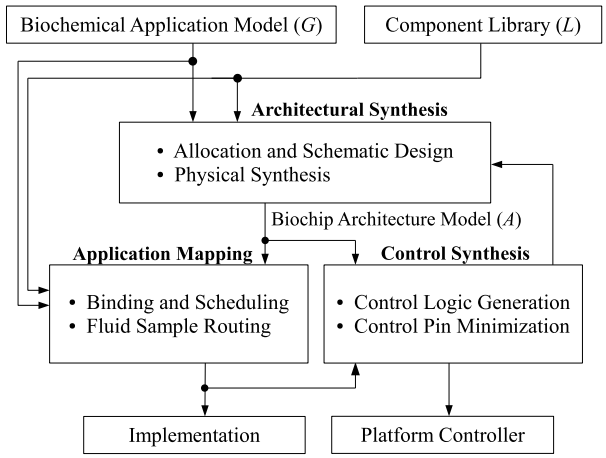


Figure 1.6: Design Methodology

the digital biochips [78]), see box "Biochemical Application Model" in Figure 1.6. We have also developed simulators and editors based on these models [62, 70].

- *Application Mapping*

  Using our proposed models, we address the problem of mapping the biochemical application onto the mVLSI biochip [59, 58], see box "Application Mapping" in Figure 1.6. We propose a constraint programming (CP) [45] framework which, given a biochemical application and a biochip architecture, determines an optimal solution (in terms of application completion time) for the binding and scheduling of the biochemical operations onto the given biochip, without considering routing. We also propose a binding and scheduling heuristic that takes the fluidic routing and channel contention into account, while aiming to generate an optimized mapping. To the best of our knowledge, this is the first time that an application mapping framework is being proposed for the mVLSI biochips.

- *Architectural Synthesis*

  We propose a top-down architectural synthesis methodology for the mVLSI microfluidic biochips [61, 57]. Given a biochemical application, a microfluidic component library and the chip area, the architectural synthesis (as shown in Figure 1.6) consists of the following three steps: (i) *allocation* of components from a given library, and performing the *schematic design* in order to generate the *netlist*, the biochip (ii) *flow layer physical synthesis*, i.e., deciding the placement of the microfluidic components on the chip and performing routing of the microfluidic flow channels on the available routing layers creating component interconnections and the (iii) *control layer physical synthesis*, i.e., deciding the placement of control pins and routing the control channels in order to connect the valves to the control pins. To the best of our knowledge, this is the first time that the architectural synthesis framework for these chips is being proposed.

- *Control Synthesis*

  We propose a top-down control synthesis framework for implementing biochemical applications on mVLSI biochips [65, 64]. As shown in Figure 1.6, given a biochip architecture and the mapping implementation of a biochemical application on the biochip architecture, control synthesis consists of the following two steps: (i) *control logic generation* and, (ii) *control pin count minimization*. Control logic generation means determining which valves need to be opened or closed, in what sequence and for how long, in order to execute the application on the chip. We utilize the output of control logic generation step and perform the control pin count minimization by sharing the control pins between multiple valves. To the best

of our knowledge, this is the first time an approach for the control logic generation for the mVLSI biochips is being proposed.

Different objective functions can be used for these design tasks. Minimizing the application completion time is a useful objective function since it minimizes the effects of environmental variations on the executing application and is also directly relevant for the clinical diagnostics and environmental monitoring applications. Minimizing the chip design area (e.g., to fit it under a microscope) or to minimize the number of control pins (to minimize the macro-assembly surrounding the chip) are also realistic objective functions. Each chapter in the thesis mentions its targeted objective function.

In addition to the above mentioned design tasks, we also address the problem of maximizing the throughput of cell culture microfluidic biochips [63]. These chips can simultaneously perform multiple experiments and their throughput is defined as the number of non-repeating experiments performed on the chip. We propose a strategy to maximize the number of experiments, saving cost both in terms of time and money (the cell culture experiments are extremely expensive to perform and it takes many days to complete one experiment).

The modeling and synthesis approach proposed is aimed at facilitating programmability and automation, reducing human effort and minimizing the design cycle time. The target is also to decouple the development of complex bioassays from the chip design and implementation process, allowing users to focus on applications. The automated design flow is expected to be an enabler for the biochip domain in the same manner as it has been for the electronic ICs in the last three decades.

1.5 Thesis Overview

The thesis is organized in seven chapters. A brief summary of these chapters is as follows:

Chapter 2 presents the details of the proposed models. It starts off by presenting some basic concepts related to the mVLSI biochip architecture, followed up by the details of the proposed component and architecture models. The application model used is also introduced in this chapter.

Chapter 3 describes the application mapping problem. It discusses the previous state-of-the-art and our contribution to this design task. A constraint programming-based optimal solution approach as well as a List Scheduling-based

binding and scheduling heuristic are described here in detail. The proposed algorithms are evaluated and the results are presented.

Chapter 4 provides the details of the mVLSI architectural synthesis problem. The problem is formulated after discussing the prior work at the start of the chapter. All the design tasks are explained in detail and then our synthesis strategy is presented. The synthesis process involves component allocation, design schematic generation, and the physical synthesis (placement and routing) of the chip. The proposed strategy is experimentally evaluated using real-life as well as synthetic benchmarks.

Chapter 5 presents the control synthesis problem in detail. Previous research is discussed at the start followed by the problem formulation. Our algorithm generates the control logic needed to execute the application and uses a Tabu Search-based optimization in order to minimize the control pin count. The approach is evaluated and the results are presented.

Chapter 6 is dedicated to the cell culture microfluidic biochips. It starts by explaining the architecture of the cell culture biochips and our modeling framework. The problem of throughput maximization for cell culture biochips is formulated using a detailed motivational example. This is followed by a discussion of our proposed solution strategy and the experimental evaluation.

The thesis is summed up by presenting the conclusions and the future work options in **Chapter 7**.

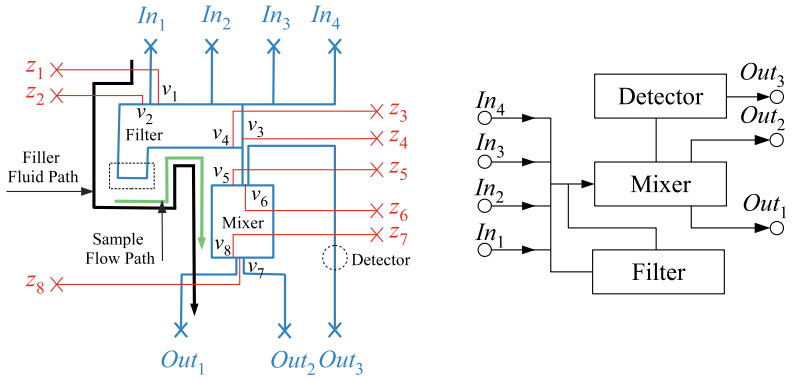
Chapter 2

System Model

We propose a topology graph-based system-level model of a biochip architecture, that is independent of the underlying biochip implementation technology. We also propose a dual-layer component model and have created a microfluidic component library.

2.1 Biochip Architecture Model

Figure 2.1a shows the schematic view of a flow-based biochip with 4 input ports and 3 output ports, 1 mixer, 1 filter, 1 detector and 8 control pins (shown in red). Figure 2.1b shows the functional view of the same chip. All fluid samples inside the chip occupy a fixed unit length (or a multiple of it) on the flow channel, i.e., the fluid samples have discretized volumes. Unit length samples are obtained by a process called *metering*, carried out by transporting the sample between two valves that are a fixed length apart [85]. In general, the chip is filled with a filler fluid (e.g., immiscible oil) and the fluid samples are emulsified in the filler fluid. As emulsions, the samples do not touch the channel walls directly (preventing cross-contamination) and can be moved over long channel lengths of any shape while retaining their content [85].



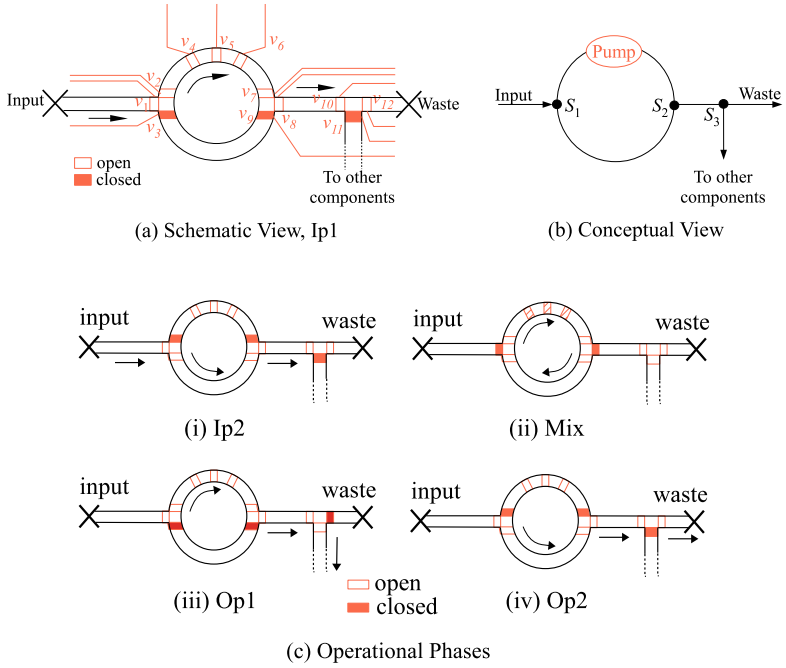
(a) Biochip: Schematic View (b) Biochip: Functional View

Figure 2.1: Flow-Based Biochip Architecture

In order to make a fluid sample flow on the chip (e.g., *Filter* to the *Mixer* in Figure 2.1a):

- The point of *fluid sample origin* (*Filter*) needs to be connected to a pump (on-chip or off-chip) for generating the flow. All chip input ports are generally equipped with an off-chip pump and the filler fluid reservoirs. As shown in Figure 2.1a, the closest pump from the *Filter* is the off-chip pump connected to the input port In_1. We term the flow starting point as the *source* (In_1 in this case).
- The *fluid sample destination point* (*Mixer*) needs to be connected to a fluidic output port (*sink*, e.g., Out_1).
- A path for the fluid flow needs to be established from the source to the sink using the microfluidic valves.
- The desired flow (*Filter* to *Mixer*) can then be achieved by activating the pump.

For the *Filter* to *Mixer* flow in Figure 2.1a, the path is established by closing the valve set v_1, v_3, v_6 and v_7, while the valve set v_2, v_4, v_5 and v_8 is kept open (the path is shown in black in Figure 2.1a). The entire path already contains the filler fluid and the sample emulsified in the filler fluid is now present inside the *Filter*. A pumping action at the source (In_1) then creates a filler fluid flow towards the sink (Out_1). The emulsified sample flows with the filler fluid from the *Filter* towards the *Mixer*. The pumping action is stopped once the



(a) Schematic View, Ip1 (b) Conceptual View

(c) Operational Phases

Figure 2.2: Microfluidic Mixer

fluid sample reaches its destination (the green path in Figure 2.1a shows the flow of the sample). While the sample flows from the *Filter* to the *Mixer*, the established path (including the source, sink points) is reserved and cannot be used for any other flows.

2.1.1 Component Model

Consider the pneumatic mixer [25] in Figure 2.2a which is implemented using nine microfluidic valves, v_1 to v_9. Figure 2.2b shows the conceptual view of the same mixer. The valve set $\{v_4, v_5, v_6\}$ acts as an on-chip pump. The valve set $\{v_1, v_2, v_3\}$ is termed as switch S_1 and the valve set $\{v_7, v_8, v_9\}$ as switch S_2. The two switches facilitate the inputs and outputs, and the pump is used to perform the mixing. The mixer output can either be sent to the waste or to the other components in the chip using the switch S_3, as shown in Figure 2.2a.

The mixer has five operational phases. The first two phases represent the input of two fluid samples that need to be mixed, followed by the mixing phase. The

mixed sample is then transported out of the mixer in the last two phases. For the first fluidic input (phase Ip1, depicted in Figure 2.2a), valves v_1, v_2, v_7 and v_8 are opened (together with v_4, v_5, v_6), the pump at the *Input* is activated and the liquid fills in the upper half of the mixer.

Note that the fluid samples that are to be mixed do not need to occupy the full channel length from the *Input* to the upper half of the mixer. Rather each sample occupies a certain length on the flow channel. As described in the previous section, the process of measuring the length of each fluid sample is called *metering* and is carried out by transporting the sample between two valves that are a fixed length apart [85].

In Figure 2.2a, the mixer output is connected to a waste outlet making a closed loop (for the filler fluid to flow in) from the input to the waste outlet. The filler fluid flows in from the input, goes through the mixer and into the waste outlet. The emulsified sample flows with the filler fluid and reaches the mixer. Once the top half is filled, the valves v_7 and v_2 close, stopping the filler fluid flow and blocking the fluid sample in the upper half of the mixer. Since we know the flow rate (mm/s) and the sample volume (in mm, measured in terms of length through *metering*), the time until the mixer gets filled can be easily calculated. Therefore, an optical feedback is not necessary in order to activate the valves.

In the next phase Ip2, the second fluid sample fills the lower half of the mixer (Figure 2.2c(i)). Once both halves are filled, the mixer input and output valves (v_1 and v_8) are closed while valves v_2, v_3, v_7, v_9 are opened and the mixing operation is initiated (Figure 2.2c(ii)). Valve set $\{v_4, v_5, v_6\}$ acts as a peristaltic pump. Closing valve v_4 inserts some pressure on the fluid inside the mixer, closing valve v_5 creates further pressure, then as valve v_6 is closed valve v_4 is opened again. This forces the liquid to rotate clockwise in the mixer. The valves are closed and opened in a sequence such that the liquid rotates at a certain speed accomplishing the mixing operation. Next, in phase Op1 (Figure 2.2c(iii)), half of the mixed sample is pushed out of the mixer towards the rest of the chip and in Op2 (Figure 2.2c(iv)), the other half is transported to the waste.

Using pressurized microfluidic valves in the control layer is the most commonly utilized control method for the flow-based biochips. However, microfluidic components equipped with alternate control techniques (e.g., electro-osmotic, electrokinetic) have also been developed [49]. In order to have a unified design methodology covering several underlying technologies, it is imperative to model the component implementation technology details separately from its operational capabilities.

We propose a dual-layer component modeling framework, consisting of a *flow layer model* and a *control layer model*. The flow layer model $(\mathcal{P}, C, H)$ of each

Table 2.1: Mixer: Control Layer Model

Phase	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
1. Ip1	0	0	1	0	0	0	0	0	1
2. Ip2	0	1	0	0	0	0	1	0	0
3. Mix	1	0	0	Mix	Mix	Mix	0	1	0
4. Op1	0	0	1	0	0	0	0	0	1
5. Op2	0	1	0	0	0	0	1	0	0

component $\mathcal{M}$ is characterized by a set of operational phases $\mathcal{P}$, execution time C and the component geometrical dimensions H. The control layer model captures the valve actuation details required for the on-chip execution of all operational phases of a component. For example, Table 2.1 presents the control layer model of a pneumatic mixer, as presented in Figure 2.2, whose flow layer model is characterized by the first row in Table 2.2. In Table 2.1, the valve activation for each phase is shown, '0' representing an open and '1' a closed valve. The status 'Mix' shown for the valve set $\{v_4,\ v_5,\ v_6\}$ on line 4 of Table 2.1 represents the mixing step in which these valves are opened and closed in a specific sequence to achieve mixing. Microfluidic platforms are equipped with a controller that manages all on-chip control, i.e., issuing signals to on-chip components for executing a biochemical application, performing data acquisition and signal processing operations [49]. The control layer model of a component contains all the details required by the biochip controller.

Table 2.2 shows the component model library $\mathcal{L} = \mathcal{M}(\mathcal{P},\ C.\ H)$ of eight commonly utilized microfluidic components [49, 21]. The geometrical dimensions H are given as length×width and are scaled, with a unit length being equal to 150μm, i.e., a length of 10 in Table 2.2 corresponds to 1500μm. Storage unit dimensions are for a storage with 8 reservoir channels and the multiplexer dimensions are for a 1-to-8 or 8-to-1 multiplexer. Multiplexers and their usage is discussed in detail in Chapter 6. The different operational phases listed for a component may or may not be executable in parallel depending on how the component is implemented, e.g., the mixer presented here has only one input port to receive both the input fluids, thus only one input phase can be activated at a time.

2.1.2 Architecture Model

The research carried out for modeling the microfluidic architecture has been focused on the device-level physical models [43, 73]. We propose a system-level model based on a topology graph in order to capture the biochip architecture.

Table 2.2: Component Library ($\mathcal{L}$): Flow Layer Model

Component	**Phases** (P)	**Exec. Time** (C)	H
Mixer	Ip1/ Ip2/ **Mix**/ Op1/ Op2	0.5 s	30×30
Filter	Ip/ **Filter**/ Op1/ Op2	20 s	120×30
Detector	Ip/ **Detect**/ op	5 s	20×20
Separator	Ip1/ Ip2/ **Separate**/ Op1/ Op2	140 s	70×20
Heater	Ip/ **Heat**/ Op	20°C/s	40×15
Metering	Ip/ **Met**/ Op1/ Op2	-	30×15
Multiplexer	Ip or Op	-	30×10
Storage	Ip or Op	-	90×30

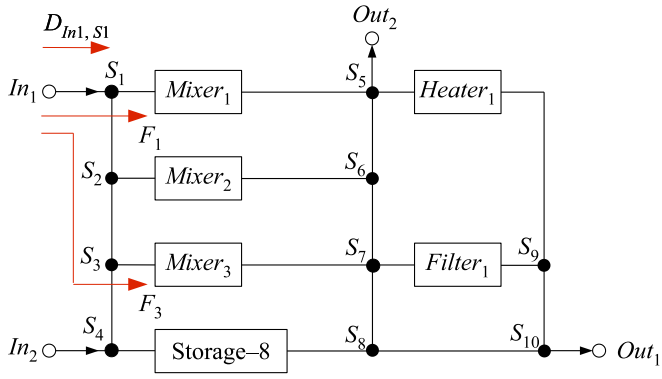


Figure 2.3: Biochip Architecture

Consider the biochip architecture shown in Figure 2.3. The chip has two inputs, two outputs and is equipped with three mixers, one heater, one filter and eight storage reservoirs, i.e., the component 'Storage-8' contains eight reservoirs, Res_1–Res_8. The biochip architecture is modeled as a topology graph $\mathcal{A} = (\mathcal{N}, \mathcal{S}, \mathcal{D}, \mathcal{F}, \mathcal{K}, c)$, where $\mathcal{N}$ is a finite set of vertices, $\mathcal{S}$ is a subset of $\mathcal{N}$, $\mathcal{S} \subseteq \mathcal{N}$, $\mathcal{D}$ is a finite set of directed edges, $\mathcal{F}$ is a finite set of flow paths and $\mathcal{K}$ is a finite set of routing constraints. A vertex $N \in \mathcal{N}$ has two distinguished types: a vertex $S \in \mathcal{S}$ represents a switch (e.g., S_1 in Figure 2.3), whereas a vertex $\mathcal{M} \in \mathcal{N}$, $\notin \mathcal{S}$, represents a component or an input/output node (e.g., $Mixer_1$ and In_1, respectively, in Figure 2.3). A directed edge $D_{i,j} \in \mathcal{D}$ represents a directed communication channel from the vertex N_i to vertex N_j, with N_i, $N_j \in \mathcal{N}$ (e.g., D_{In_1,S_1} represents a directed link from vertex In_1 to vertex S_1). A flow path, F_i $\in \mathcal{F}$, is a subset of two or more directed edges of $\mathcal{D}$, $F_i \subseteq \mathcal{D}$, $|F_i| > 1$, representing a directed communication link between any two vertices $\in \mathcal{N}$ using a chain of directed edges of $\mathcal{D}$ (e.g., $F_{In_1,Mixer_1} = (D_{In_1,S_1}, D_{S_1,Mixer_1})$ represents a

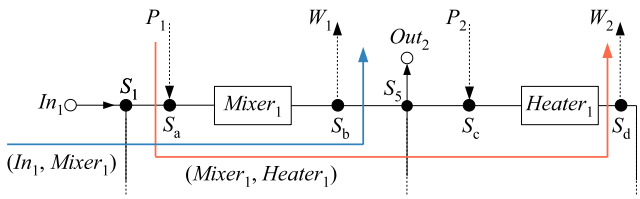


Figure 2.4: Biochip Architecture

directed link from vertex In_1 to vertex $Mixer_1$). A routing constraint, $K_i \in \mathcal{K}$, is a set of flow paths that are mutually exclusive with the flow path $F_i \in \mathcal{F}$, i.e., none of the flow paths in the set can be activated in parallel. For example, F_1 and F_3 in Figure 2.3 are mutually exclusive as they share the vertices In_1 and S_1. The function $c(y)$, where y is either a directed edge $D \in \mathcal{D}$ or a flow path $F_i \in \mathcal{F}$, represents its routing latency (time required by a fluid sample to traverse y).

The set of flow paths $\mathcal{F}$ is the set of permissible flow routes on the biochip. The flow path starts from a point of fluid sample origin and ends at the fluid sample destination point. These flow paths are specified by the biochip designer but can be easily extracted from the chip architecture as well, if all pump locations, filler fluid inlet and waste outlet locations are known. As discussed earlier, in order for a route to be permissible on the chip (e.g., Figure 2.3., flow from chip input In_1 to the mixer $Mixer_1$, $F_{In_1,Mixer_1} = (D_{In_1,S_1}, D_{S_1,Mixer_1})$), the point of flow origin (In_1) needs to be connected to a pump (and a filler fluid reservoir) and the point of destination ($Mixer_1$) needs to be connected to a waste output.

In Figure 2.4, $Mixer_1$ and $Heater_1$ are provided with external filler fluid inlets (P_1, P_2) connected to external pumps and waste outlets (W_1, W_2). An external fluid sample can now be brought into $Mixer_1$ (In_1 to $Mixer_1$) using In_1 as the source point for the flow and W_1 as the sink point, forming the complete source-sink path (In_1, S_1, S_a, $Mixer_1$, S_b, W_1), depicted in blue in Figure 2.4. Similarly, moving the fluid sample out of $Mixer_1$ and in to $Heater_1$ can be done using the oil inlet P_1 as the source point and the waste outlet W_2 as the sink point. The flow path is from $Mixer_1$ to $Heater_1$ (this is the path that the fluid sample travels) and the complete source-sink path (closed loop paths for the filler fluid that allows the fluid sample to flow on the chip) is (P_1, S_a, $Mixer_1$, S_b, S_5, S_c, $Heater_1$, S_d, W_2), shown in red in Figure 2.4.

Providing such an independent connection (of filler fluid inlets, pumps and waste outlets) to every vertex requires a huge macro-assembly around the chip. Therefore, only a limited number of vertices are provided with these connections, limiting the number of allowed flows on the chip. The allowed flows are captured

Table 2.3: Flow Path Set ($\mathcal{F}$)

$F_1 = (In_1, S_1, Mixer_1)$, 2 s
$F_2 = (In_1, S_1, S_2, Mixer_2)$, 2.5 s
$F_3 = (In_1, S_1, S_2, S_3, Mixer_3)$, 3 s
$F_4 = (In_2, S_4, S_3, S_2, S_1, Mixer_1)$, 3.5 s
$F_5 = (In_2, S_4, S_3, S_2, Mixer_2)$, 3 s
$F_6 = (In_2, S_4, S_3, Mixer_3)$, 2.5 s
$F_{7-x} = (In_1, S_1, S_2, S_3, S_4, Storage\text{-}8)$, 3.5 s
$F_{8-x} = (In_2, S_4, Storage\text{-}8)$, 2 s
$F_9 = (Mixer_1, S_5, Out_2)$, 2 s
$F_{10} = (Mixer_1, S_5, Heater_1)$, 2 s
$F_{11} = (Mixer_1, S_5, S_6, S_7, Filter_1)$, 3 s
$F_{12-x} = (Mixer_1, S_5, S_6, S_7, S_8, Storage\text{-}8)$, 3.5 s
$F_{13} = (Mixer_1, S_5, S_6, S_7, S_8, S_{10}, Out_1)$, 4 s
$F_{14} = (Mixer_2, S_6, S_5, Out_2)$, 2.5 s
$F_{15} = (Mixer_2, S_6, S_5, Heater_1)$, 2.5 s
$F_{16} = (Mixer_2, S_6, S_7, Filter_1)$, 2.5 s
$F_{17-x} = (Mixer_2, S_6, S_7, S_8, Storage\text{-}8)$, 3 s
$F_{18} = (Mixer_2, S_6, S_7, S_8, S_{10}, Out_1)$, 3.5 s
$F_{19} = (Mixer_3, S_7, S_6, S_5, Out_2$, 3 s
$F_{20} = (Mixer_3, S_7, S_6, S_5, Heater_1)$, 3 s
$F_{21} = (Mixer_3, S_7, Filter_1)$, 2 s
$F_{22-x} = (Mixer_3, S_7, S_8, Storage\text{-}8)$, 2.5 s
$F_{23} = (Mixer_3, S_7, S_8, S_{10}, Out_1)$, 3 s
$F_{24-x} = (Storage\text{-}8, S_4, S_3, S_2, S_1, Mixer_1)$, 3.5 s
$F_{25-x} = (Storage\text{-}8, S_4, S_3, S_2, Mixer_2$, 3 s
$F_{26-x} = (Storage\text{-}8, S_4, S_3, Mixer_3)$, 2.5 s
$F_{27-x} = (Storage\text{-}8, S_8, S_7, S_6, S_5, Heater_1)$, 3.5 s
$F_{28-x} = (Storage\text{-}8, S_8, S_7, Filter_1)$, 2.5 s
$F_{29-x} = (Storage\text{-}8, S_8, S_{10}, Out_1)$, 2.5 s
$F_{30-x} = (Heater_1, S_9, S_{10}, S_8, Storage\text{-}8)$, 3 s
$F_{31} = (Heater_1, S_9, S_{10}, Out_1)$, 2.5 s
$F_{32-x} = (Filter_1, S_9, S_{10}, S_8, Storage\text{-}8)$, 3 s
$F_{33} = (Filter_1, S_9, S_{10}, Out_1)$, 2.5 s

by the set of flow paths $\mathcal{F}$. Table 2.3 shows a possible flow path set (permissible route set), $\mathcal{F}$, for the biochip given in Figure 2.3. A shorter representation (using the vertices traversed in the flow path) is chosen for clarity, for example, the flow path $F_{In_1,Mixer_1} = (D_{In_1,S_1}, D_{S_1,Mixer_1})$ is represented as $F_1 = (In_1, S_1, Mixer_1)$. Also, note that each flow path involving the storage reservoir (e.g., F_{7-x}) represents a set of eight flow paths (F_{7-1} to F_{7-8}), i.e., one for each of the eight storage reservoirs. Each route (flow path) has an associated control

layer model that contains the details required for its utilization, i.e., the switch sequence and the pump activation details.

The fluid transport latencies, $c(F)$, associated with each flow path are also listed in Table 2.3. For calculating the latencies, we abstract away from absolute fluid volumes and utilize the concept of a unit fluid volume instead (captured by *metering* as explained in Section 2.1.1). Each fluidic I/O (input/output phase of a component) is characterized by a volume weight w_v, which is used to calculate the transport latency of a certain flow path when utilized for that specific fluidic I/O. Similarly, each component also has an associated capacity weight w_c, representing its volume capacity. For this example, we assume a volume weight of one for all fluidic I/Os. The capacity weight of all microfluidic components is assumed to be the same as its number of input phases, e.g., a mixer has two input phases, therefore it has a capacity weight equal to two. Also, a fluid with volume weight one occupies a fixed channel length w_l on the chip. In the thesis, we assume this channel length to be equal to 10 mm.

The latencies for the flow paths have been calculated using a typical flow rate of 10 mm/s [49] and the chip dimensions of 5 mm between any two network vertices, N_i and N_j (termed as a segment), with N_i, $N_j \in \mathcal{N}$. For example, $F_1 = (In_1, S_1, Mixer_1)$ traverses two segments, i.e., In_1 to S_1 and S_1 to $Mixer_1$, thus a total channel length of 10 mm. With a flow rate of 10 mm/s, a fluid with volume weight one (occupying a total channel length of 10 mm) would have a total latency of 2 seconds from the time the fluid tip enters from In_1 till the fluid tail disappears into the mixer $Mixer_1$.

Analogous to a circuit-switched network, when a flow path gets activated, the entire route (from the source to the sink) is reserved until the completion of the fluid transfer. This imposes routing constraints on the chip. All those flow paths in the set $\mathcal{F}$ that have a network vertex N_i in common in their source-sink paths are considered as mutually exclusive, i.e., the routes represented by these flow paths can only be utilized in a serialized fashion. For example in Figure 2.3, $F_{In_1,Mixer_1}$ and $F_{In_1,Mixer_2}$ are mutually exclusive as they share the vertices In_1 and S_1. The routing constraints associated with the flow path set in Table 2.3 are shown in Table 2.4. The first row in the routing constraints ($K_1 : (F_2, F_3, F_4, F_7, F_{24})$) shows that F_1 cannot be executed in parallel with F_2, F_3, F_4, F_7 and F_{24}.

Since fluid samples are expendable and cannot be reused limitless number of times (unlike the operands in computers), the fluid volumes need to be managed inside the chip. Researchers have proposed methods for carefully distributing the liquid volume, preventing overflow and underflow of the fluid samples [20]. We assume that the designer does this beforehand while designing the biochemical application, ensuring that both overflow and underflow are avoided.

Table 2.4: Routing Constraints $\mathcal{K}$

$K_1 : (F_2, F_3, F_4, F_7, F_{24})$
$K_2 : (F_1, F_3, F_4, F_5, F_7, F_{24}, F_{25})$
$K_3 : (F_1, F_2, F_4, F_5, F_6, F_7, F_{24}, F_{25}, F_{26})$
$K_4 : (F_1, F_2, F_3, F_5, F_6, F_7, F_8, F_{24}, F_{25}, F_{26})$
$K_5 : (F_2, F_3, F_4, F_6, F_7, F_8, F_{24}, F_{25}, F_{26}, F_{27})$
$K_6 : (F_3, F_4, F_5, F_7, F_8, F_{24}, F_{25}, F_{26})$
$K_{7-x} : (F_1, F_2, F_3, F_4, F_5, F_6, F_8\ , F_{24}, F_{25}, F_{26})$
...
$K_{33} : (F_{13}, F_{18}, F_{23}, F_{29}, F_{30}, F_{31}, F_{32})$

2.2 Biochemical Application Model

Biochemical applications have traditionally been described through a sequence of steps given in free-flowing English-language text. Such descriptions are often ambiguous and incomplete and are not adequate for automation of biochemical protocols. Researchers have proposed standardizing programming languages in order to express biochemical applications [22].

We model a biochemical application using a sequencing graph [88]. Real-life assays can be converted to this model using [22]. The graph $\mathcal{G}(\mathcal{O}, \mathcal{E})$ is directed, acyclic and polar (i.e., there is a *source* vertex that has no predecessors and a *sink* vertex that has no successors). Each vertex $O_i \in \mathcal{O}$ represents an operation that can be bound to a component using a binding function $\mathcal{B} : \mathcal{O} \rightarrow \mathcal{M}$. Each vertex has an associated weight $C_i^{M_j}$, which denotes the execution time required for the operation O_i to be completed on component M_j. The execution times provided in Table 2.2 are of the actual functional phase (given in bold in the table, e.g., **Mix**). These execution times are taken as the typical execution times for the said component types, i.e., typical mixing time is 0.5 s but a biochemical application description may specify a longer time (e.g., 5 s) if required for a particular operation. This value does not include the time required to fetch the input fluids or to remove the output fluids from the component. The input/output (I/O) phases are dependent on the chip architecture and are thus captured by the set of flow paths $\mathcal{F}$ in the biochip architecture model $\mathcal{A}$. The edge set $\mathcal{E}$ models the dependency constraints in the assay, i.e., an edge $e_{i,j} \in \mathcal{E}$ from O_i to O_j indicates that the output of O_i is the input of O_j. All inputs need to arrive before an operation can be activated. We assume that the biochemical application has been correctly designed, such that all operations will have the correct volume of liquid available for their execution. Figure 2.5 shows an example of a biochemical application model which has seven mixing operations (O_1–O_4, O_6, O_7, O_{10}), one filtration operation (O_9) and two heating operations

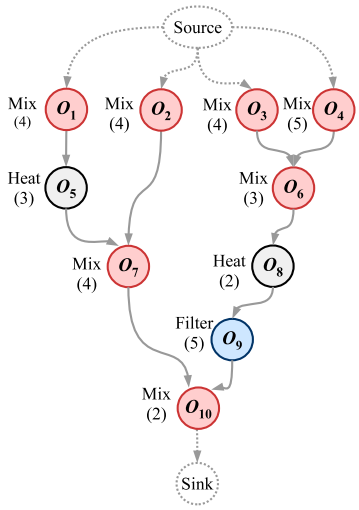


Figure 2.5: Application Model

(O_5, O_8). The execution times for the operations are given in Figure 2.5 (the parameter below the operation type).

2.3 Benchmarks

We use both real-life and synthetic benchmarks in order to evaluate our proposed design methodology.

2.3.1 Real-Life benchmarks

We consider three real-life benchmarks that are also used for the digital biochips.

2.3.1.1 Polymerase Chain Reaction (PCR)

The first real-life assay is the PCR (polymerase chain reaction) mixing stage that has 7 mixing operations and is used in DNA amplification. In PCR, several thermal cycles are used to replicate a piece of DNA, creating thousands of copies. This protocol is very useful when the available DNA sample is not in sufficient

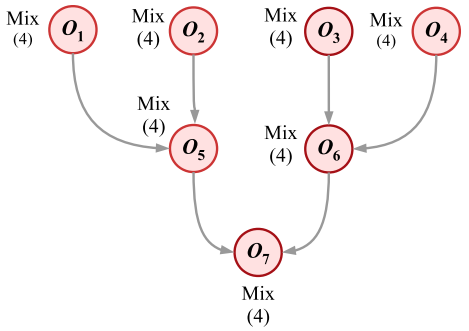


Figure 2.6: Polymerase Chain Reaction (PCR)

enough quantity for performing the analysis. The first step of the polymerase chain reaction consists of seven mixing operations, denoted in Figure 2.6 by O_1 to O_7. The output of this stage undergoes a series of thermal cycles for performing DNA amplification [44].

2.3.1.2 In-vitro Diagnostics (IVD)

Multiplexed IVD (in-vitro diagnostics) has a total of 12 operations. Figure 2.7 describes the protocol for an in-vitro diagnostics assay (IVD) in which the level

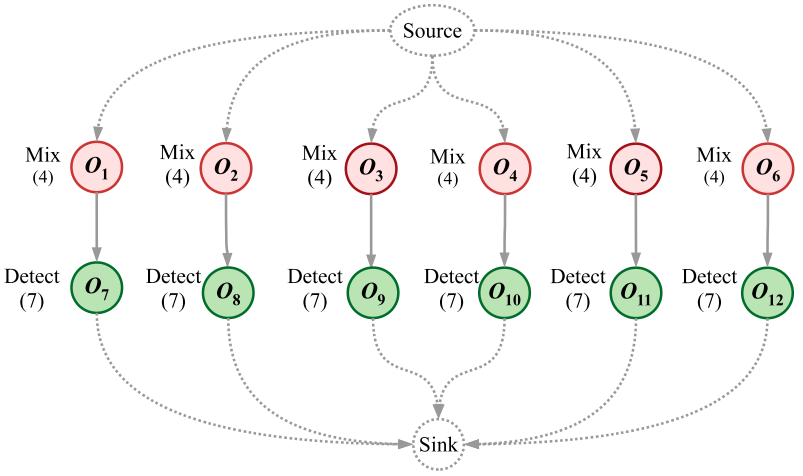


Figure 2.7: In-Vitro Diagnostics (IVD)

of different metabolites in human physiological fluids are measured. The assays requires the input of samples (urine, plasma, and serum), reagents (glucose oxidase, lactate oxidase) and buffer substance. The level of glucose and oxidase are measured for each type of physiological fluid using the detection operations.

2.3.1.3 Colorimetric Protein Assay (CPA)

The application graph in Figure 2.8 describes a protein assay which is used for determining the concentration of a certain protein in a solution. The procedure causes a reaction between the protein of interest and a dye. The concentration of the protein is determined by measuring the absorbance of a particular wavelength in the resulted substance. The protocol consists of 55 microfluidic operations and uses three types of liquids: physiological fluid (sample containing the protein), Coomassie Brilliant Blue G-250 dye as reagent and NaOH as buffer substance. Before being mixed with the dye, the sample is first diluted with the NaOH buffer. Dilution is represented as mixing in the application graph. The protocol finishes with detection operations, in which the protein concentration for the resultant solution is measured. The letter "S" in the application graph represents the Source, which means that the input comes from the off-chip reservoirs.

2.3.2 Synthetic benchmarks

We consider five different synthetic benchmarks. The benchmark applications are composed of 10, 20, 30, 40 and 50 operations. Figure 2.9 – 2.13 present the synthetic benchmarks.

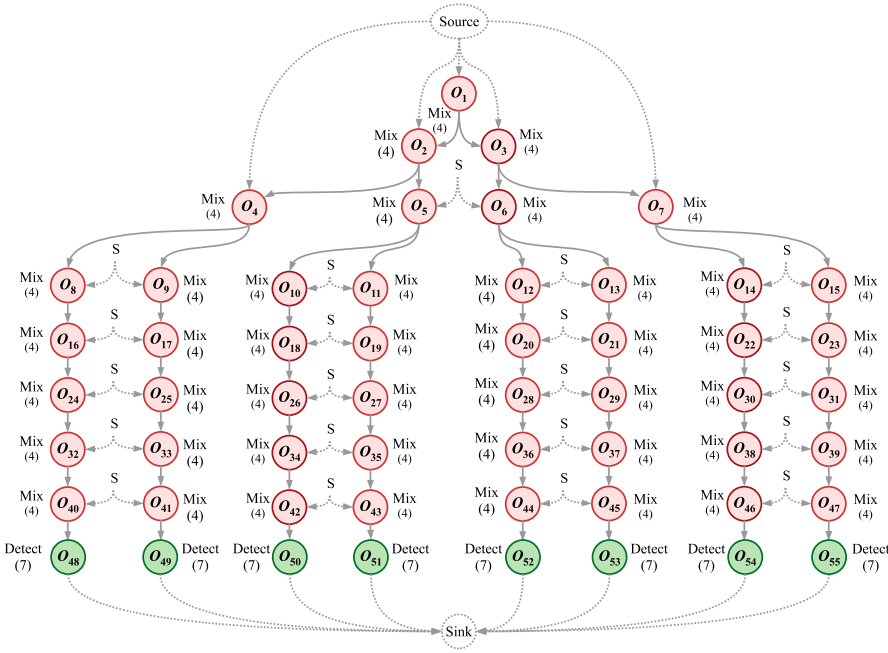


Figure 2.8: Colorimetric Protein Assay (CPA)

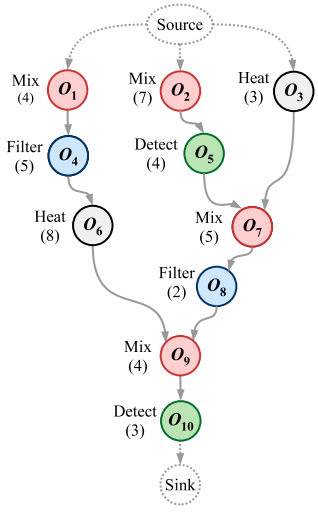


Figure 2.9: Synthetic Benchmark - 10 Operations

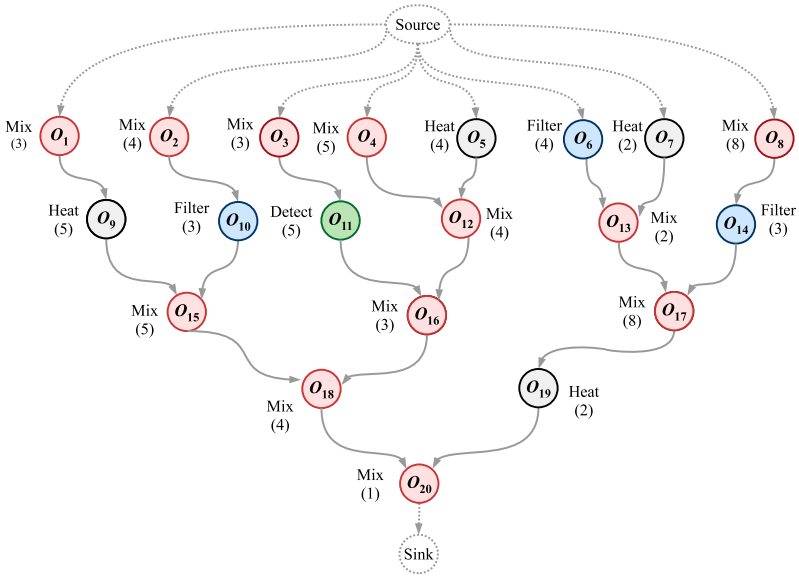


Figure 2.10: Synthetic Benchmark - 20 Operations

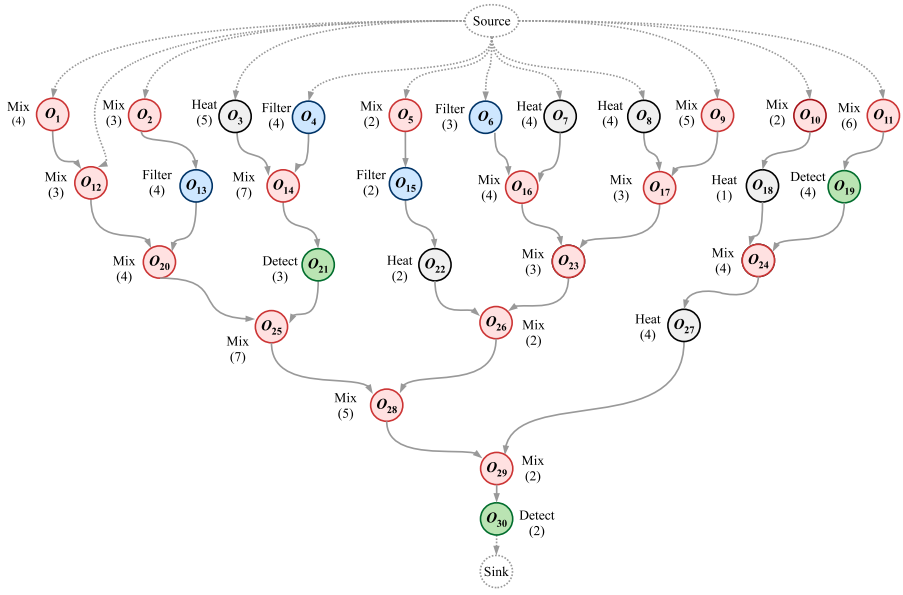


Figure 2.11: Synthetic Benchmark - 30 Operations

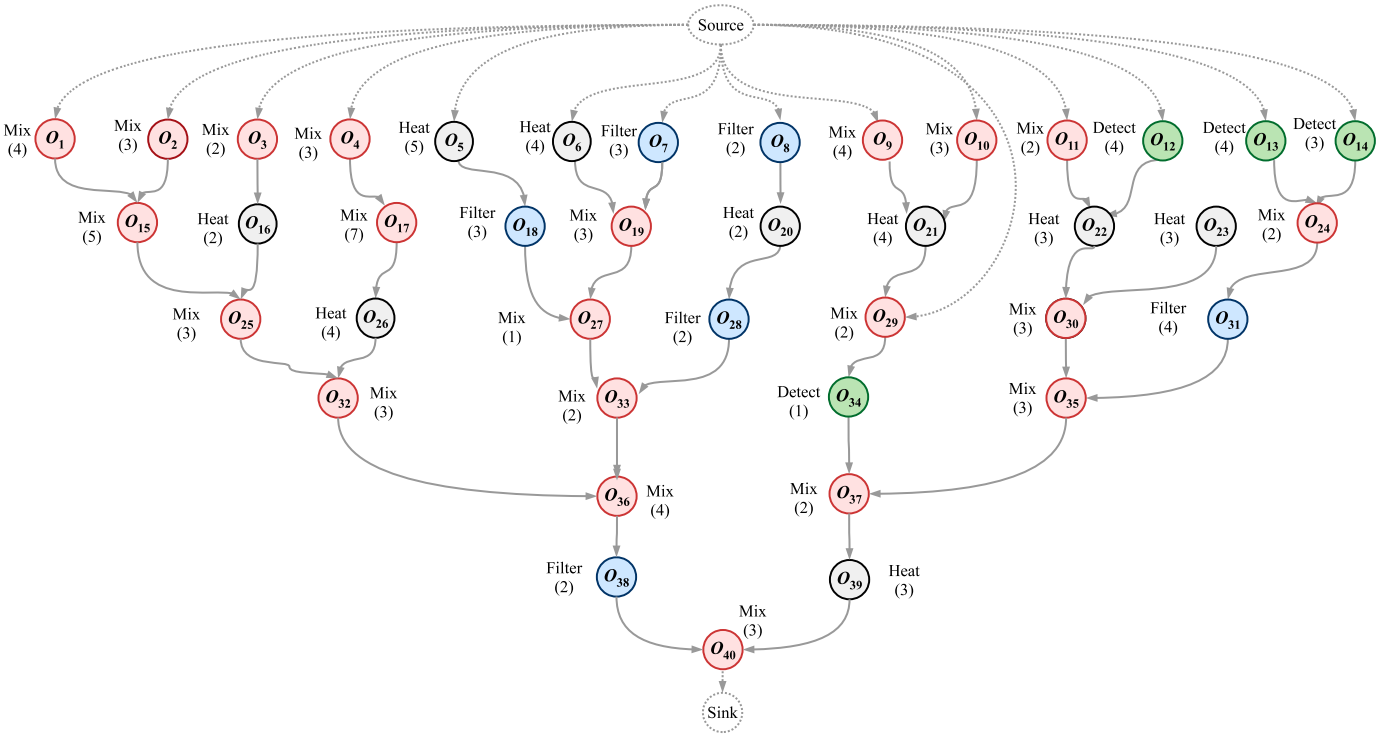


Figure 2.12: Synthetic Benchmark - 40 Operations

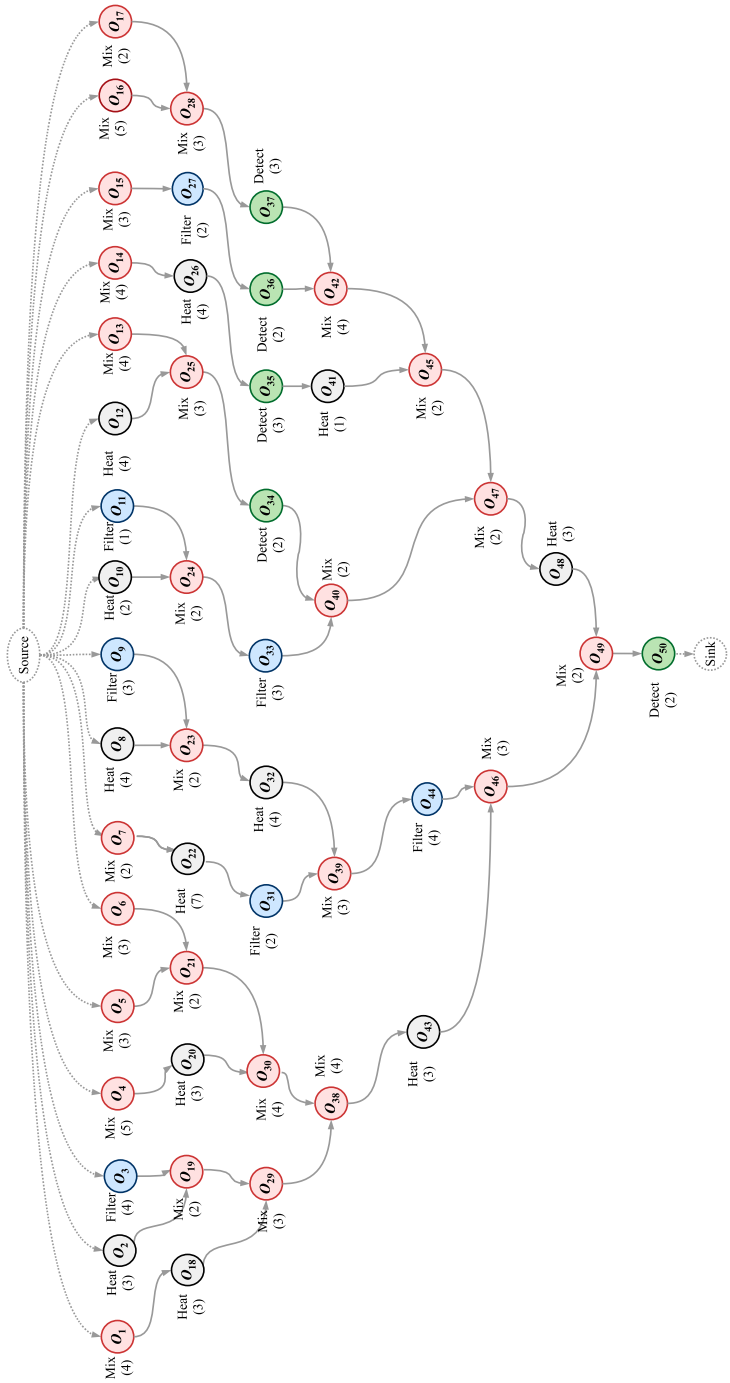


Figure 2.13: Synthetic Benchmark - 50 Operations

2.4 Summary

In this chapter we have presented our proposed biochip architecture and component models. The model used for the biochemical application has also been introduced. We have also given the details of the benchmarks considered in this thesis. Using these models, in the next chapter, we propose our approach for mapping the biochemical applications onto the mVLSI biochips and evaluate the approach using the mentioned benchmark applications.

Chapter 3

Application Mapping

The block diagram of our proposed design methodology is shown in Figure 1.6. In this chapter, we focus on the "Application Mapping" block. It takes the biochemical application model and the models of the biochip architecture and the biochip components as input. As output, it generates the implementation $\Psi < \mathcal{B}, \mathcal{X} >$ which contains the binding and scheduling details of the operations as well as the fluid routing information.

3.1 Related Work

Currently, researchers manually map the applications to the valves of the chip using some custom interface (analogous to exposure of gate-level details) [81]. The manual process is quite tedious and needs to be repeated every time a change is made either to the chip architecture or the biochemical application. For larger chips and applications, the process can easily result in inefficient application mappings.

Researchers have proposed significant work on top-down synthesis techniques for droplet-based biochips [24]. However, as discussed in Chapter 1, these techniques are not applicable to the flow-based chips and, to the best of our knowl-

edge, no automated application mapping approach has been proposed so far for the flow-based biochips.

3.2 Contribution

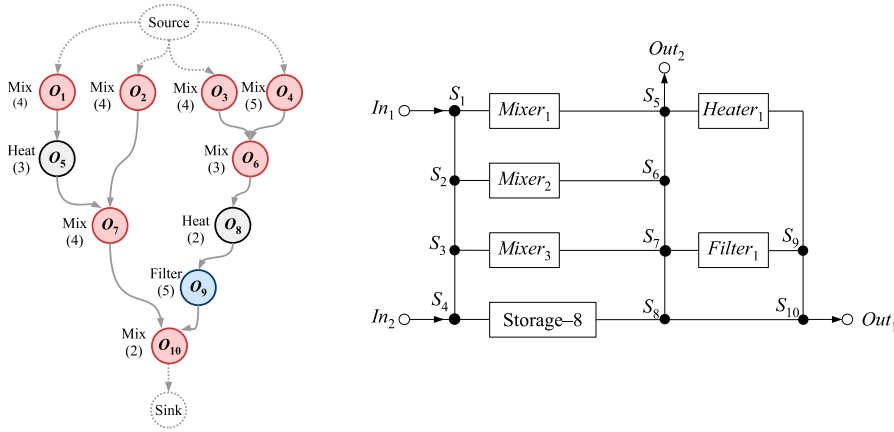
Using the models proposed in the previous chapter, we focus on the problem of mapping a biochemical application, modeled as a sequencing graph (capturing the operations and their dependency constraints), onto a given biochip architecture. We propose a constraint programming (CP) [45] framework which, given a biochemical application and a biochip architecture, determines an optimal solution (in terms of application completion time) for the binding and scheduling of the biochemical operations onto the given biochip. CP makes it possible to specify the resource and timing constraints, and to capture the application binding and scheduling within the same framework. Using the CP formulation, a solver then searches for the optimal solution.

In microfluidic biochips, routing latencies are comparable to the operation execution times, thus having a considerable influence on the schedule. The CP-based solutions, although optimal, require a large computation time when more complex chips are introduced and fluidic routing is included inside the implementation. We propose a List Scheduling (LS)-based binding and scheduling heuristic that also takes the fluidic routing and channel contention into account, while aiming to generate an implementation that minimizes the application completion time. The heuristic produces good quality solutions in small time. We evaluate the proposed framework by synthesizing real-life case studies as well as synthetic benchmarks.

Next section discusses the design tasks involved in the biochip synthesis. The targeted problem is formulated at the end of this section. The proposed CP-based synthesis approach is presented in Section 3.4 and the LS-based approach in Section 3.5. We evaluate our framework in Section 3.6

3.3 Application Mapping

Mapping the application onto the architecture involves *binding* of operations onto the allocated components, *scheduling* the operations and performing the required fluidic *routing*. This section explains these design tasks using the biochemical application in Figure 3.1a and the biochip architecture given in Figure 3.1b. The architecture is modeled as described in Section 2.1. Thus, the



(a) Application Graph (b) Biochip Architecture

Figure 3.1: Illustrative Example

allocated components are captured by the vertex set $\mathcal{M}$, $\mathcal{M} \in \mathcal{N}$, in the architecture model $\mathcal{A}$. Table 3.1 shows the set $\mathcal{M}$ for the biochip given in Figure 3.1b. The component placement and interconnections are also given, and are captured by the remaining elements of the topology graph $\mathcal{A}$ modeling the architecture, as discussed in Section 2.1. Table 2.3 shows the flow path set (permissible route set), $\mathcal{F}$, for the biochip given in Figure 3.1b. The routing constraints (as discussed in Section 2.1.2) extracted from the set are shown in Table 2.4.

Figure 3.2 shows the schedule for executing the biochemical application in Figure 3.1a on the biochip architecture in Figure 3.1b. The schedule is represented as a Gantt chart, where, we represent the operations and fluid routing phases as rectangles, with their lengths corresponding to their execution duration. Each operation is placed in a separate row. During the binding step, each vertex O_i, $O_i \in \mathcal{O}$, representing a biochemical operation in the application model in Fig-

Table 3.1: Allocated Components ($\mathcal{M}$)

Function	Units	Notations
Input port	2	In_1, In_2
Output port	2	Out_1, Out_2
Mixer	3	$Mixer_1$, $Mixer_2$, $Mixer_3$
Heater	1	$Heater_1$
Filter	1	$Filter_1$
Storage Reservoir	8	Res_1–Res_8

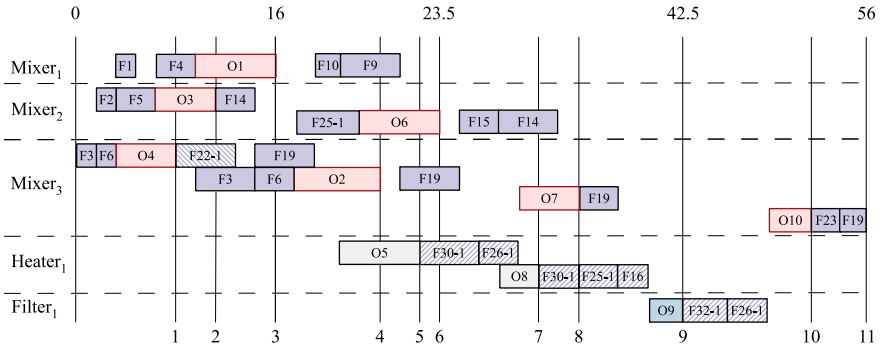


Figure 3.2: Schedule

ure 3.1a is bound to an available component M_j, i.e., $\mathcal{B}(O_i) = M_j$. For example, the mixing operation O_1 in the application model in Figure 3.1a is bound to the component $Mixer_1$ as shown in Figure 3.2. Since the fluid transport latencies in microfluidic chips are comparable to the operation execution times, fluid routing also needs to be considered during the synthesis phase. This means that the binding function must also capture the binding of the edge set $\mathcal{E} \in \mathcal{G}$ to an available route. The available route can be a flow path, $F \in \mathcal{F}$, or a collection of flow paths called a *composite route*. A composite route is used if the source and destination components are such that no direct flow path exists between them.

A scheduling strategy is needed to efficiently execute the biochemical operations on the chip components, while considering the dependency and resource constraints captured by the biochemical application and the biochip architecture models, respectively. In Figure 3.2, operation O_6 bound to $Mixer_2$ starts immediately after all its predecessors (O_3, O_4) are complete and the input fluids have been routed to $Mixer_2$. It starts at $t_{O_6}^{start} = 20.5$ s and takes 3 s, finishing at time $t_{O_6}^{finish} = 23.5$ s.

Together with the set of operations $\mathcal{O} \in \mathcal{G}$ given in the application model, the edge set $\mathcal{E} \in \mathcal{G}$ also needs to be scheduled on the chip, while taking the routing constraints into account. Before scheduling the edge, the implementation needs to evaluate if a flow path $F \in \mathcal{F}$ is sufficient to bind the edge, or if a collection of flow paths (composite route) is needed. For example, the edge $e_{6,8}$, modeling the transport of the output of $O_6(Mixer_2)$ (operation O_6 bound to component $Mixer_2$) to $O_8(Heater_1)$, can be directly bound to the flow path F_{15} (Table 2.3). The edge $e_{5,7}$ models the output of $O_5(Heater_1)$ being transported to $O_7(Mixer_3)$. However, there is no flow path $F \in \mathcal{F}$ that connects $Heater_1$ to

$Mixer_3$. Therefore, a composite route (consisting of a collection of flow paths) needs to be generated. The edge $e_{5,7}$ is bound to the composite route (F_{30-1}, F_{26-1}) as shown in Figure 3.2.

During the scheduling phase, the storage requirement analysis needs to be performed as well. This means that after completion of an operation, a decision on whether the output fluid (analogous to the operand) should be moved to the storage reservoir or not, needs to be made.

3.3.1 Problem Formulation

The problem addressed here can be formulated as follows: Given (1) a biochemical application modeled as a sequencing graph $\mathcal{G}$, (2) a biochip architecture modeled as a topology graph $\mathcal{A}$, and (3) a characterized component library $\mathcal{L}$, we are interested in synthesizing an implementation Ψ that minimizes the application completion time while satisfying the dependency, resource and routing constraints. Synthesizing an implementation $\Psi = < \mathcal{B}, \mathcal{X} >$ means deciding on (1) the binding $\mathcal{B}$ of each operation $O_i \in \mathcal{O}$ to a component $M_j \in \mathcal{M}$, and each edge $e_{i,j} \in \mathcal{E}$ to a flow path $F_i \in \mathcal{F}$ (or to a composite flow path generated by the implementation), and (2) the schedule $\mathcal{X}$ of the operations and the edges, which contains the start time t^{start} of each operation O_i and edge $e_{i,j}$ on its corresponding component and (composite) flow path.

3.4 Constraint Programming-Based Strategy

The problem can be considered equivalent to the resource constrained scheduling problem with non-uniform weights, which is NP-complete [83, 60]. CP offers very good performance for such problems [45]. Typically, a problem defined in CP has three primary elements: (1) a set of *variables* capturing the system, (2) a set of *finite domains* of the values for these variables and (3) a set of *constraints* imposed on these variables. The *solution* of such a problem is the assignment of values to all variables from their respective domains such that all constraints are satisfied. If an *optimal* solution is desired, then a cost function also needs to be defined in terms of the variables. The solver then tries to find the optimal solution in terms of the cost function that satisfies all constraints.

Our CP-based implementation generates optimal solutions for the binding and scheduling of operations onto the biochip architecture. This approach ignores the fluidic routing, which we address in Section 3.5. Figure 3.3 shows the optimal

schedule for executing the application in Figure 3.1a onto the architecture in Figure 3.1b. We have used the constraint programming environment Gecode [71] for our implementation.

3.4.1 Finite Domain Variables

We use the following primary finite domain variables (FDV) to model the binding and scheduling of all biochemical operations:

- $t_{O_i}^{start} :: \{0..\infty\}$ defines the start time of operation O_i. For example, operation O_5 in Figure 3.3 has the start time $t_{O_5}^{start} = 4$ s.
- $M_i :: \{0..|\mathcal{M}|-1\}$ defines the resource (component) to which the operation O_i is bound where $|\mathcal{M}|$ is the total number of components on the chip. For example, in Figure 3.3 $|\mathcal{M}| = 5$. Each resource is assigned a unique numeric identifier (ID) and the range of the identifiers is $0..|\mathcal{M}|-1$, i.e., 0..4 for the current example. For Figure 3.3, ID of $Mixer_1$ is 0, $Mixer_2$ is 1, $Mixer_3$ is 2, $Heater_1$ is 3 and $Filter_1$ is 4. So for operation O_5, $M_5 = 3$, which is the ID of $Heater_1$.
- $\delta_{\mathcal{G}} :: \{0..\infty\}$ defines the cost function (application completion time). Application completion time is the end-to-end time taken by the application to complete its execution, i.e., from the start time of the first executed operation to the finish time of the last executed operation. For example in Figure 3.3, the application completion time is $\delta_{\mathcal{G}} = 17$ s, which is the finish time of the last executed operation O_{10}.

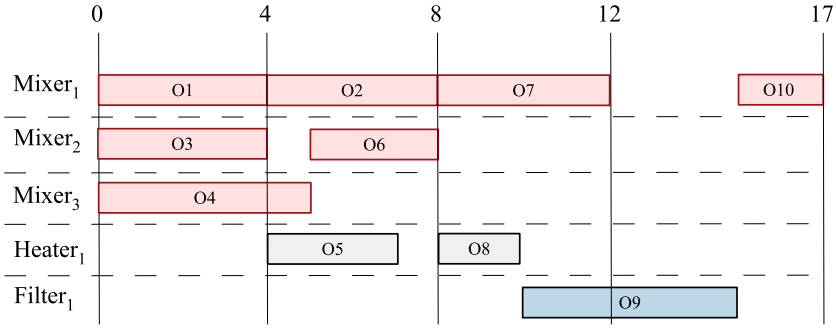


Figure 3.3: Optimal Schedule

Secondary FDVs are introduced, where needed, in order to implement the constraints.

3.4.2 Resource Binding Constraints

An operation can only be bound to a component which is capable of executing it, i.e., a mix operation must be bound to one of the mixers and not to any other component, e.g., heaters. Based on the type of the operations, we constrain the domain of the FDV M_i in order to exclude the forbidden components. Each operation is treated as a tuple (O_i, α_i), where $O_i \in \mathcal{O}$ is an operation and $\alpha_i \subseteq \mathcal{M}$ is a set of available components capable of performing the operation. A binding must respect,

$$M_i \in \alpha_i, \quad \forall i \in \{1..|\mathcal{O}|\} \tag{3.1}$$

where $|\mathcal{O}|$ defines the total number of operations in the application graph $\mathcal{G}$, e.g., application given in Figure 3.1a has 10 operations. For operation O_1 (a mix operation), $\alpha_1 = \{0, 1, 2\}$ which represents the IDs of the three mixers. For O_1 in Figure 3.3, $M_1 = 0$ (ID for $Mixer_1$) which is a member of the set α_1, thus satisfying the resource binding constraint.

3.4.3 Resource Sharing Constraints

Operations bound to the same component (e.g., O_3 and O_6 are bound to $Mixer_2$ in Figure 3.3) must not overlap in time. We use three disjunctive constraints in order to implement this for all possible combinations of operations.

$$\forall i, \forall (j > i)$$

$$(t_{O_i}^{start} + C_i \leq t_{O_j}^{start}) \vee (t_{O_j}^{start} + C_j \leq t_{O_i}^{start}) \vee (M_i \neq M_j), \quad \forall (i,j) \in \{1..|\mathcal{O}|\} \tag{3.2}$$

where C_i represents the execution time of operation O_i. We consider two operations (O_i, O_j) at a time. At least, one of the three constraints given in the above equation needs to be true in order to ensure that the resources are correctly shared.

The first constraint, $(t_{O_i}^{start} + C_i \leq t_{O_j}^{start})$, means that the finish time of operation O_i should be less than or equal to the start time of operation O_j. For

example for (O_1, O_2) in Figure 3.3, the finish time for O_1 (4 s) is equal to the start time of O_2, thus the constraint is satisfied. If the first constraint is not satisfied (consider (O_7, O_2) in Figure 3.3, the finish time for O_7 (12 s) is not less than or equal to the start time of O_2 (4 s)), then the second constraint, ($t_{O_j}^{start} + C_j \leq t_{O_i}^{start}$), is considered. For ($O_7$, O_2) in Figure 3.3, the finish time for O_2 (8 s) is equal to the start time of O_7 (8 s), thus the constraint is satisfied for this pair of operations. The third constraint, $M_i \neq M_j$, is considered if both the first and second constraint are not satisfied. Consider (O_1, O_3), the finish time for O_1 (4 s) is not less than or equal to the start time of O_3 (0 s) and the finish time for O_3 (4 s) is also not less than or equal to the start time of O_1 (0 s). In this case, the third constraint must be true. Here, $M_1 = 0$ ($Mixer_1$) and $M_3 = 1$ ($Mixer_2$), satisfying the resource sharing constraint.

3.4.4 Precedence Constraints

During scheduling, an operation must not start executing until its predecessor has completed its execution, e.g., in Figure 3.1a, O_6 must finish before O_8 can start. The following constraint is added for each required precedence:

$$t_{O_i}^{start} + C_i \leq t_{O_j}^{start} \tag{3.3}$$

where operation O_i is a predecessor of operation O_j.

3.4.5 Cost Function

We are interested in a solution that minimizes the application completion time (cost function $\delta_{\mathcal{G}}$). We constrain the cost function to be greater than or equal to the finish time of the operation O_i, i.e.,

$$t_{O_i}^{start} + C_i \leq \delta_{\mathcal{G}}, \quad \forall i \in \{1..|\mathcal{O}|\} \tag{3.4}$$

$$minimize \quad \delta_{\mathcal{G}} \tag{3.5}$$

Minimization of the $\delta_{\mathcal{G}}$ generates the optimal solution, which satisfies all the imposed constraints. In order to reduce the search space and speed up the result generation, we place an upper and lower bound on $\delta_{\mathcal{G}}$. The upper bound is the sum of execution times of all operations as that is the largest possible application completion time. For the example in Figure 3.1a, the upper bound is 36. The lower bound is equal to the duration of the critical path in the given application graph $\mathcal{G}$. Critical path is defined as the path in the application

graph, going from the source nodes to the sink nodes, that has the largest duration. For example in Figure 3.1a, the critical path consists of the nodes $\{O_4, O_6, O_8, O_9, O_{10}\}$ and has the duration equal to 17 s. The lower bound value (*lower_bound*) is provided as an input to our CP framework.

$$\delta_{\mathcal{G}} \leq \sum C_i, \quad \forall i \in \{1..|\mathcal{O}|\} \tag{3.6}$$

$$\delta_{\mathcal{G}} \geq lower\_bound \tag{3.7}$$

3.5 List Scheduling-Based Strategy

The manufacturing technology for the flow-based chips has advanced faster than Moore's law [39], resulting in ever increasing complexity for these chips. Although CP-based approach generates optimal solutions but as the problem becomes larger and more complex, the CP-based approach becomes computationally too intensive.

We propose a heuristic approach to solve the problem in a computationally efficient manner. Together with the operation binding and scheduling, our heuristic approach also considers the fluidic routing and channel contention.

The requirement of scheduling the routing together with the task operations, while satisfying the routing constraints, makes the problem analogous to the communication contention aware scheduling in parallel computing systems. We utilize the well-known *List Scheduling Algorithm* (LS) [56] and extend it with contention awareness [74] by also scheduling the edges ($\mathcal{E}$ in $\mathcal{G}$) onto the communication channels during the synthesis process.

The schedule shown in Figure 3.2 has been generated using our LS-based synthesis algorithm shown in Figure 3.4. The operations of the biochemical application are topologically sorted based on the dependency constraints. At each control step, operations are evaluated and the ready ones are found (the operations whose predecessor operations have been completed). Each new control step marks an operation event generation, with the operation event being defined as the completion of a scheduled operation. The list of ready operations is prioritized using the *urgency criteria*. The urgency of an operation is specified by the length of the longest path from the operation to the sink, i.e., summing up the execution weights of the vertices and the latency times for the edges. An average latency of 3 s is considered per edge based on the biochip architecture given in Figure 3.1b. An average is used since it is unclear on which flow paths the edges would be bound, unless binding of all operations has been completed

BiochipSynthesis($\mathcal{G}$, $\mathcal{A}$, $\mathcal{L}$)

```
1  Initialize < B, B^o, X > to φ
2  while <all operations and edges are not scheduled> do
3    // Phase I: Bind Operations and Edges
4    while <all possible ready operations are not bound> do
5      B^o = BindOperations(G, A)
6      B^o = BindStorage(B^o, B, X)
7      B^o = GenRouteAndBindEdges(B^o, A, L)
8    end while
9    B = Record(B^o)
10   // Phase II: Schedule Operations and Edges
11   while <an operation event does not occur> do
12     X = ScheduleOperationsAndEdges(B, X, A)
13     Advance time to next event
14   end while
15 end while
16 return  Ψ = < B, X >
```

Figure 3.4: Synthesis Algorithm for Flow-Based Biochips

(defining the source and destination component for each edge). The urgency value for O_1 in Figure 3.1a is 25. If the number of ready operations exceeds the number of available resources, the most urgent operations (having higher urgency value) are scheduled and the remaining ones are deferred.

We perform the implementation synthesis in two phases. In Phase-I, we start off by binding the ready operations to the available resources (line 5). The algorithm tries all possible bindings and chooses the one that produces the shortest completion time for that operation. For example after control step 5, both $Mixer_1$ and $Mixer_3$ are available as the mixing operation O_7 is released, i.e., both its predecessor operations O_2 and O_5 have been completed. As shown in Figure 3.2, O_2 was bound to $Mixer_3$ and its output is still inside the mixer unit when O_7 is released (i.e., its output has not been moved to the storage unit). The algorithm binds O_7 to $Mixer_3$ preventing the routing delay that would have occurred had O_7 been bound to $Mixer_1$.

Next, we evaluate if a reservoir is required to store the output of the operations that finished in the previous control step (line 6) and if so, bind it to a particular reservoir. A storage reservoir is utilized only (1) if the component to which the previous operation was bound is needed for performing another operation and (2) the successor of the previous operation is not scheduled during the current control step. For example in Figure 3.2, O_4 (bound to $Mixer_3$) was completed in control step 1. Its successor (O_6) is not scheduled in control step 2 and

$Mixer_3$ is needed to perform O_2. Based on the above given criteria, output of O_4 is bound to the storage reservoir Res_1 at the start of control step 2.

Next we generate the routes (single flow path or composite route) for performing these operations and bind the edges to the generated routes (line 7). We start by fetching the phase information of the components (e.g., $Mixer_3$ for O_4) from the library $\mathcal{L}$ (Table 2.2) and bind the I/O phases (Ip1, Ip2, Op1, Op2) to the corresponding generated routes (e.g., F_3 for Ip1 of $Mixer_3$). If a route is not found, the operation is deferred. The algorithm jumps back to line 5 in order to modify the binding for the current control step ($\mathcal{B}^o$), by binding a low priority operation that was earlier deferred because of lack of resources. Once the binding has been finalized, it is recorded into the binding information $\mathcal{B}$ (line 9). More details about route generation are given in following subsection.

In Phase-II (lines 11–14), we generate the schedule for the operations and the edges bound in Phase-I. Starting from the input edges associated with the operation of the highest priority (based on the urgency criteria, O_4 in this case), we start scheduling the edges one by one (here the first one is F_3). In order to reduce the schedule length, we try to schedule as many bound edges as possible in parallel. All the ready edges that do not violate the routing constraints (listed in Table 2.4) can be utilized in parallel. An edge is considered a ready edge if it does not violate any inter- or intra-operation dependency constraints. The inter-operation dependency constraints are given by the application model $\mathcal{G}$ (e.g., O_3 and O_4 need to complete before O_6). The intra-operation dependency constraints means that the inputs of an operation and the operation itself need to be completed before its outputs can be issued (e.g., F_3, F_6 and the mixing operation in $Mixer_3$ need to complete before F_{22-1} and F_{19} can be scheduled). Multiple inputs/ outputs for the same operation (F_3, F_6 for $Mixer_3$) are independent of each other and can be parallelized if the routing constraints permit.

Figure 3.5 shows the schedule for the first control step (Figure 3.2 shows the complete schedule). Since O_4 (bound on $Mixer_3$) has the highest priority, F_3 is the first bound edge that is scheduled. None of the other edges (bound to flow paths) in the ready set $< F_6,\ F_2,\ F_5,\ F_1,\ F_4 >$ can be scheduled in parallel with F_3 because of the routing constraints given in Table 2.4, thus F_3 is scheduled alone as shown in Figure 3.5. Once all possible edges and operations are scheduled (only F_3 in this case), we advance time to the next event (operation event marking completion of an operation, or an edge event representing completion of an edge execution) (line 13). An operation event triggers a new control step and the algorithm switches back to Phase-I, whereas an edge event means that the next edge needs to be scheduled. Completion of F_3 thus triggers an edge event. We schedule F_6 (next edge in the highest priority operation O_4) and try to optimize the schedule again (F_2 can be scheduled in parallel with F_6). The operations are scheduled as soon as all their input edges

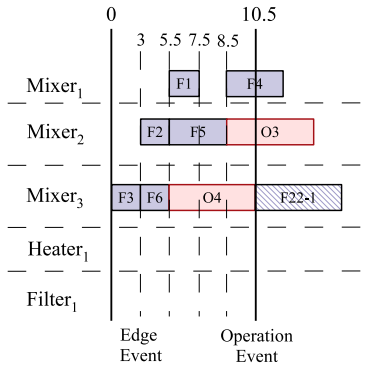


Figure 3.5: Edge and Operation Events

have been executed (e.g., O_4 after F_3 and F_6). When an operation finishes (e.g., O_4 on $Mixer_3$ in control step 1) it triggers an operation event and the algorithm jumps back to Phase-I.

The process is repeated until all operations/ edges are scheduled. The implementation Ψ, consisting of binding $\mathcal{B}$ and scheduling $\mathcal{X}$ information, is then returned (line 16).

3.5.1 Route Generation

GenRouteAndBindEdges (line 7) is used to generate the fluid route from the selected source to the selected destination (e.g., from In_1 to $Mixer_1$). If a flow path exists, then it binds the edge to that flow path and returns the binding information. If the selected operation has the selected source and destination such that no flow path exists to route the fluid from the source to the destination (e.g., $Heater_1 \rightarrow Mixer_3$, control step 5–7 in Figure 3.2), then the algorithm searches for a composite route, i.e., a route linking the desired source and destination using more than one flow path.

If multiple composite routes exist, the shortest one (in terms of cumulative latencies) is selected. For ($Heater_1 \rightarrow Mixer_3$), the composite route ($Heater_1 \rightarrow Res_1$, $Res_1 \rightarrow Mixer_3$) is selected. Note that this requires all intermediate destinations (Res_1 — reservoir 1 in the storage) to be available.

If no direct route is available and no composite route can be specified as well (e.g., intermediate destinations not available), then the operation is deferred.

Phase-I (lines 4–8) is repeated again to see if any of the other low priority operations (that were not scheduled earlier because of resource constraints) can now be scheduled instead. Once all possible operations and edges are bound, the algorithm switches to Phase-II.

3.6 Experimental Evaluation

We evaluate our proposed CP- and LS-based approaches by synthesizing real life assays as well as synthetic benchmarks onto different biochip architectures. The CP framework was implemented in Gecode constraint programming environment [71] and the LS-based algorithm was implemented in C++, running on Lenovo T400s ThinkPad with Core 2 Duo Processors at 2.53 GHz and 4 GB of RAM.

Table 3.2 shows our experimental results for the CP-based synthesis approach. Column 1 presents the application and column 2 shows the list of allocated components in the following format (Mixers, Heaters, Filters, Detectors). Column 3 presents the optimal application completion time $\delta_{\mathcal{G}}$ obtained using CP. We compare the results to our list scheduling based approach by ignoring the fluidic routing in the LS-based approach as well. Column 4 shows the application completion time generated using the LS-based approach. Column 5 presents the time taken by CP to generate the solution and column 6 gives the time taken by the LS-based approach.

The first real-life assay is the PCR (polymerase chain reaction) mixing stage. We synthesize the assay on three different biochip architectures varying the number of mixers. Each mixing operation is considered to have an execution time of 4 s on the mixing units used in the biochips. As shown in Table 3.2, both CP and LS generate optimal solutions.

Next real-life assay is the multiplexed IVD (in-vitro diagnostics). As given in Table 3.2, IVD is synthesized onto three different biochip architectures. Each mixing operation is considered to have an execution time of 4 s and the detection operation 7 s. Increasing the number of components reduces $\delta_{\mathcal{G}}$ for IVD from 25 s to 11 s.

The example application (EA) given in Figure 3.1a is used as a synthetic benchmark. We vary the number of resources and generate optimal values of $\delta_{\mathcal{G}}$. The CP framework facilitates design space exploration enabling biochip designers to take early design decisions. For example in Table 3.2, the result (row 8 and row 9) shows that increasing the number of mixers from 3 to 4 and the heaters

Table 3.2: Experimental Results: CP-Based Synthesis

Appl.	Allocated Components	$\delta_{\mathcal{G}-CP}$	$\delta_{\mathcal{G}-LS}$	CP Exec. Time	LS Exec. Time
PCR	(2, 0, 0, 0)	16 s	16 s	0.2 s	< 0.1 s
	(3, 0, 0, 0)	16 s	16 s	0.2 s	< 0.1 s
	(4, 0, 0, 0)	12 s	12 s	0.2 s	< 0.1 s
IVD	(2, 0, 0, 2)	25 s	25 s	2.6	< 0.1 s
	(5, 0, 0, 5)	18 s	18 s	38 min 28 s	< 0.1 s
	(6, 0, 0, 6)	11 s	11 s	1 min 38 s	< 0.1 s
EA	(2, 1, 1, 0)	19 s	19 s	0.3 s	< 0.1 s
	(3, 1, 1, 0)	17 s	17 s	0.4 s	< 0.1 s
	(4, 2, 1, 0)	17 s	17 s	0.5 s	< 0.1 s

from 1 to 2 does not result in any improvement in $\delta_{\mathcal{G}}$, allowing the designer to use lesser components and reduce chip area.

As given in Table 3.2, LS-based approach also generates the same solution as the CP-based approach. The CP-based approach, however, takes longer than the LS-based approach as it spends time in ensuring the solution optimality. Even in these small example cases, the CP-based approach can take as much as 38 minutes longer than the LS-based approach (row 5 in Table 3.2). For larger, more complex problems the CP-based approach is expected to take an even longer computation time. Therefore, we switch to our LS-based approach for generating good quality solutions in small time.

Table 3.3 shows our experimental results for the LS-based synthesis approach. These results are for the approach discussed in Section 3.5, which also considers fluidic routing and the number of chip I/O ports used to dispense or remove fluid samples from the chip. Columns 2 and 3 give the number of input and output ports. Column 4 presents the details of the architectures considered, in terms of (Mixers, Heaters, Filters, Detectors) and column 5 gives the number of storage units utilized. The last column in the table presents the completion time, in seconds.

We synthesize the PCR assay on three different biochip architectures varying the number of I/O ports and mixer units. As given in Table 3.3, increasing the number of mixers and the I/O ports directly influences the application completion time $\delta_{\mathcal{G}-LS}$ bringing it down to 27.5 s in the second architecture. Increasing the resources further only results in more routing constraints, taking the completion time up to 28.2 s.

Table 3.3: Real-Life Assays: LS-Based Synthesis

Application	I/p Ports	O/p Ports	Allocated Components	Storage Units	$\delta_{\mathcal{G}-LS}$
PCR	2	2	(2, 0, 0, 0)	3	30.3 s
	3	3	(3, 0, 0, 0)	1	27.5 s
	4	4	(4, 0, 0, 0)	3	28.2 s
IVD	2	2	(2, 0, 0, 2)	4	31.3 s
	6	6	(6, 0, 0, 6)	0	13 s

Multiplexed IVD is synthesized onto two different biochip architectures. As we can see, increasing the number of mixers and detectors that can work in parallel, from 2 to 6, brings down $\delta_{\mathcal{G}-LS}$ from 31.3 s to 13 s.

Varying the number of resources directly influences the chip area and also the schedule length. Chip area is an important parameter for the chips that need to be placed in small chambers, e.g., under microscopes for detection. Our methodology captures the design at the top level and can be utilized by the designer to evaluate their designed architectures and make design decisions at an early stage, minimizing the design cycle time and associated cost.

In a final set of experiments we have evaluated our proposed LS-based method using a set of different synthetic benchmarks. The benchmark applications are composed of 10 up to 50 operations. Table 3.4 shows the details of the architectures considered and the respective application completion times achieved. No storage units were utilized by the applications in these experiments.

As shown in Table 3.4, we only vary the number of input ports in the considered architectures and note the impact on the application completion time $\delta_{\mathcal{G}-LS}$. For the 10 node application, reducing the number of input ports from 2 to 1 has a minor influence (0.6 % increase) on $\delta_{\mathcal{G}-LS}$. However, for the 20 node application benchmark, the completion time increases by approximately 19.2 %, going from 28.1 s to 33.5 s. For the 30, 40 and 50 node applications, for the first case we consider only 2 input ports. In the second architecture, we consider the optimal number of chip input ports, i.e., providing all the inputs required from off-chip reservoirs in parallel. For the 30 node application, increasing the input ports to the optimal number produces a reduction of approximately 8 % in the completion time. For the 40 and 50 node applications, the reduction is slightly more significant, around 11.2 % and 11.5 % respectively.

There are many factors that influence the application completion time, e.g., component interconnection scheme, application operation types and their se-

Table 3.4: Synthetic Benchmarks: LS-Based Synthesis

Nodes	Input Ports	Output Ports	Allocated Components	$\delta_{\mathcal{G}-LS}$
10	2	2	(4, 2, 2, 2)	31.3 s
	1	1	(4, 2, 2, 2)	31.5 s
20	2	2	(12, 4, 3, 1)	28.1 s
	1	1	(12, 4, 3, 1)	33.5 s
30	2	1	(17, 6, 4, 3)	39 s
	12	1	(17, 6, 4, 3)	35.9 s
40	2	1	(21, 9, 6, 4)	36.4 s
	15	1	(21, 9, 6, 4)	32.3 s
50	2	1	(26, 12, 7, 5)	39.8 s
	17	1	(26, 12, 7, 5)	35.2 s
EA	2	2	(3, 1, 1, 0)	56 s
	4	1	(7, 2, 1, 0)	21.7 s

quence, operation execution times. The last two rows in Table 3.4 show the results for the example application (EA) given in Figure 3.1a. We vary the I/O ports as well as the number of components resulting in a new completion time of 21.7 s (61 % less than the previous completion time of 56 s). The models can be used to further explore the relationships between these parameters, resulting in aiding design of optimal application-specific biochip architectures.

All of the experiments presented in this section took less than 1 s of run time to complete. All benchmarks can be found in Section 2.3.

3.7 Summary

We have proposed a constraint programming (CP)-based approach that optimally synthesizes a biochemical application onto the specified biochip architecture with the application completion time minimization as the target objective. The synthesis process involves performing binding and scheduling of operations while satisfying the dependency and resource constraints. Our CP approach ignores fluid routing. We have also proposed a more computationally efficient heuristic approach that also takes into account fluidic routing (contention aware edge scheduling) together with the operation mapping. Real-life case studies and a set of synthetic benchmarks have been synthesized on different architectures for validating the proposed approach. The proposed approach is expected to reduce human effort, enabling designers to take early design decisions by being

able to evaluate their proposed architecture, minimizing the design cycle time and also facilitating programmability and automation.

The next chapter presents our top-down methodology for designing the application-specific architecture for the flow-based mVLSI biochips.

Chapter 4

Architectural Synthesis

This chapter focuses on the "Architectural Synthesis" block of our proposed design methodology (see Figure 1.6). It takes the biochemical application and the component library as input and synthesizes the application-specific biochip architecture. We start off by presenting the related work and describing our contributions. The problem is formulated in Section 4.3 and the tasks involved in architectural synthesis are explained in Section 4.4. Our solution strategy is presented in Section 4.5 and evaluated in Section 4.6.

4.1 Related Work

Designers are using manual and bottom-up methodologies to implement these chips. Microfluidic components are designed and connected together to match the steps of the desired biochemical application using technical drawing tools such as AutoCAD [3]. The placement and routing is also done manually [15] and then the chip is fabricated using soft lithography. Recent work has proposed automation techniques for the placement and routing of the control layer [21].

Manual design of application-specific mVLSI chips is very time-consuming and requires significant design effort. In order to overcome this, a general-purpose

biochip architecture has been proposed [19]. The soft lithography based fabrication process is, however, cheap and has a fast turn around time [55], pointing to having application-specific chips capable of providing higher efficiency instead of doing a general-purpose design. No scheme for the application-specific architectural synthesis has been proposed.

4.2 Contribution

We propose a top-down architectural synthesis methodology for the flow-based mVLSI biochips. Given a biochemical application modeled as a sequencing graph, a microfluidic component library and the chip area, the architectural synthesis consists of the following two steps: (i) *allocation* of components from a given library and performing the *schematic design* in order to generate the *netlist*, and the biochip (ii) *physical synthesis*, i.e., deciding the placement of the microfluidic components on the chip and performing routing of the microfluidic channels on the available routing layers creating component interconnections (for both flow and control layers).

The synthesis problem is NP-complete. We use an approach similar to High-Level Synthesis [26] for performing allocation and netlist generation. The component placement is done using Simulated Annealing and we tailor the Hadlock's algorithm [69] from the Very Large-Scale Integrated (VLSI) circuits domain for performing the microfluidic channel routing. Synthesis is done in such a way that the application completion time is minimized and the imposed constraints (e.g., resource, dependency) are satisfied.

To the best of our knowledge, this is the first time an approach for the automatic synthesis of a biochip architecture is being presented. The main contributions here are the formulation of the architectural synthesis problem and the proposed synthesis framework, which show how the well-known algorithms from the High-Level Synthesis of VLSI circuits can be tailored to tackle the mVLSI biochips.

4.3 Problem Formulation

The problem addressed here can be formulated as follows: Given a biochemical application modeled as a sequencing graph $\mathcal{G}$ and a characterized component library $\mathcal{L}$, we are interested in synthesizing a biochip architecture $\mathcal{A}$, such that the application completion time is minimized and the imposed constraints are

satisfied. The synthesis approach can handle several constraints, such as overall chip area, maximum number of components of a certain type and the number of external input and output ports. The number of external ports is also limited by the maximum number of punch holes possible on the chip under the given design rules [15].

Synthesizing an architecture $\mathcal{A}$ means deciding on (1) the allocation $\mathcal{U}$ of components from the component library $\mathcal{L}$, (2) the configuration for interconnection of these components (netlist), (3) placement $\mathcal{Z}_f$ of the components onto the chip layout area and interconnecting them by flow channel routing $\mathcal{R}_f$, and (4) placement $\mathcal{Z}_c$ of control valves and control pins on the chip and interconnecting them by control channel routing $\mathcal{R}_c$. The flow path set $\mathcal{F}$, associated latencies and the corresponding routing constraints $\mathcal{K}$ also need to be extracted from the synthesized architecture.

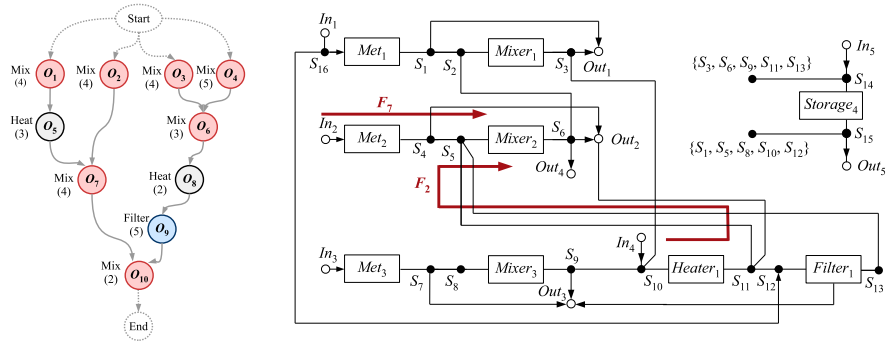
As mentioned, the objective of the problem is to minimize the application completion time under the given constraints. However, other objectives can also be handled, such as the minimization of the architecture cost under a given timing constraint. Reliability of an mVLSI biochip depends directly on the reliability of the valves (the valves can operate reliably only up to a few thousand actuations). Therefore, in order to achieve enhanced reliability, an optimization step can be added directed at balancing the load on the valves, i.e., each valve goes through approximately the same number of valve actuations during the application execution.

4.4 Biochip Architectural Synthesis

The following subsections explain the design tasks involved in the biochip synthesis using Figure 4.1 as an illustrative example. Next section presents our proposed synthesis framework for these tasks.

4.4.1 Allocation and Schematic Design

In this step, the microfluidic components required for implementing the given biochemical application $\mathcal{G}$ are allocated from the component library $\mathcal{L}$, while taking into account the imposed resource constraints. Next, based on the given application, a chip schematic is designed and the netlist is generated. For example, to implement the biochemical application from Figure 4.1a under the constraints given in Table 4.1 columns 1 and 2, we could use an allocation $\mathcal{U}$



(a) Application Graph (b) Biochip Architecture

Figure 4.1: Biochip Application and Architecture Example

as captured by the last two columns in Table 4.1. The schematic design corresponding to such an application and allocation is presented in Figure 4.1b. Note that the storage units are needed in order to save the output of a component so that it can be used at a later stage. The flow path set is also generated in this step. A flow path, as discussed in Section 2.1.2, is the path starting from the point of fluid sample origin and ending at the fluid sample destination point, e.g., $Heater_1$ to $Mixer_2$ in Figure 4.1b. Source-sink paths associated with each flow path are also defined, e.g., for the flow path $Heater_1$ to $Mixer_2$ in Figure 4.1b, the source-sink path is (In_4, S_{10}, $Heater_1$, S_{11}, S_5, $Mixer_2$, S_6, Out_2). Routing constraints are also extracted at this stage. Two flow paths, whose corresponding source-sink paths have a common vertex are mutually exclusive and need to be listed under the routing constraints, e.g., F_7 and F_2 in Figure 4.1b are mutually exclusive since they share common vertices (e.g., S_5) in their source-sink paths. Table 4.2 shows the flow path set, the source-sink set and Table 4.3 shows the routing constraints associated with the architecture in

Table 4.1: Allocated Components ($\mathcal{U}$)

Function	**Constraints**	**Allocated Units**	**Notations**
Input port	5	5	In_1 ... In_5
Output port	5	5	Out_1 ... Out_5
Mixer	3	3	$Mixer_1$ ... $Mixer_3$
Heater	2	1	$Heater_1$
Filter	1	1	$Filter_1$
Metering Units	3	3	Met_1 ... Met_3
Storage Units	4	4	$Storage_x$

Table 4.2: Flow Path Set ($\mathcal{F}$) and the Source-Sink Set

Flow Path Set	Source-Sink Set
$F_0 = (Mixer_3, S_9, S_{10}, Heater_1)$, 0.7 s	$F'_0 = (In_3, Met_3, S_7, S_8, Mixer_3, S_9, S_{10}, Heater_1, S_{11}, Out_2)$
$F_1 = (Mixer_2, S_6, S_2, Mixer_1)$, 0.4 s	$F'_1 = (In_2, Met_2, S_4, S_5, Mixer_2, S_6, S_2, Mixer_1, S_3, Out_1)$
$F_2 = (Heater_1, S_{11}, S_5, Mixer_2)$, 0.5 s	$F'_2 = (In_4, S_{10}, Heater_1, S_{11}, S_5, Mixer_2, S_6, Out_2)$
$F_3 = (Mixer_1, S_3, S_{10}, Heater_1)$, 0.6 s	$F'_3 = (In_1, S_{16}, Met_1.\ S_1, S_2, Mixer_1, S_3, S_{10}, Heater_1, S_{11}, Out_2)$
$F_4 = (Heater_1, S_{11}, S_{12}, Filter_1)$, 2.1 s	$F'_4 = (In_4, S_{10}, Heater_1, S_{11}, S_{12}, Filter_1, Out_3)$
$F_5 = (Filter_1, S_{13}, S_5, Mixer_2)$, 0.8 s	$F'_5 = (In_1, S_{16}, S_{12}, Filter_1, S_{13}, S_5, Mixer_2, S_6, Out_2)$
$F_6 = (In_1, S_{16}, Met_1, S_1, S_2, Mixer_1)$, 1.3 s	$F'_6 = (In_1, S_{16}, Met_1, S_1, S_2, Mixer_1, S_3, Out_1)$
$F_7 = (In_2, Met_2, S_4, S_5, Mixer_2)$, 1.9 s	$F'_7 = (In_2, Met_2, S_4, S_5, Mixer_2, S_6, Out_2)$
$F_8 = (In_3, Met_3, S_7, S_8, Mixer_3)$, 2.1 s	...
$F_9 = (Mixer_1, S_3, Out_1)$, 1.2 s	$F'_{28-x} = (In_1, S_{16}, S_{12}, Filter_1, S_{13}, S_{14}, Storage, S_{15}, Out_5)$
$F_{10} = (Mixer_2, S_6, Out_2)$, 0.3 s	
$F_{11} = (Mixer_3, S_9, Out_3)$, 0.6 s	
...	
$F_{28-x} = (Filter_1, S_{13}, S_{14}, Storage)$, 0.5 s	

Figure 4.1b. Additional routing constraints may be imposed during the placement and routing phases, resulting in an updated routing constraints list.

4.4.2 Physical Synthesis

In this step, the allocated components are placed on a chip layout area and the interconnections between components are routed as channels on the chip such that the application completion time is minimized. The placement and routing phases are governed by design rules (see Table 4.4) imposed by the fabrication process carried out in a standard microfluidic foundry [15, 21]. During placement, the components are treated as fixed size blocks, represented by rectangles, each having a fixed length and width (dimensions given in the component library in Table 2.2). The placement is done in such a way that all design rules are satisfied and no two components overlap on the chip.

For mVLSI-based biochips, the placement and routing phases can be divided into two stages, one for each logical layer in the chip: the flow layer and the control layer.

4.4.2.1 Flow Layer

This stage involves determining the placement of microfluidic components and the fluidic inlet/ outlet ports $\mathcal{Z}_f$ on the chip layout area, and then routing the interconnecting nets $\mathcal{R}_f$ as microfluidic flow channels. Only one layer is available for performing the flow channel routing [55]. In VLSI chips, the intersection of nets is considered a short-circuit and is thus not permitted. However, net intersection is possible in the biochip flow layer. A switch is placed at the location of the intersection so that both nets (a net represents a microfluidic channel) can be used, at different points in time, without unintended fluid mixing. Considering that only one layer is available for routing all flow channel nets, the possibility of net intersection helps in achieving 100% routability. However, net intersections cause routing constraints, resulting in longer application completion times. Figure 4.2 shows the placement and routing scheme for the flow layer of the biochip architecture shown in Figure 4.1b. The entire placement and routing shown is done in one layer.

Table 4.3: Routing Constraints ($\mathcal{K}$)

K_0 : (F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{11}, F_{12}, F_{13}, F_{14} F_{15}, F_{16}, F_{17}, F_{18}, F_{19}, F_{20}, F_{21}, F_{24}, F_{25}, F_{26}, F_{27}, F_{28})
K_1: (F_0, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{12}, F_{13}, F_{14}, F_{15}, F_{16}, F_{17}, F_{24}, F_{25}, F_{26})
...
...
K_{28-x}: (F_0, F_3, F_4, F_5, F_6, F_7, F_9, F_{11}, F_{14}, F_{15}, F_{16}, F_{17}, F_{18}, F_{19}, F_{20}, F_{21}, F_{22}, F_{23}, F_{25}, F_{26}, F_{27})

Table 4.4: Design Rules

Parameter	**Suggested Value**
Width of flow channel	100 μm
Minimum spacing between flow channels	40 μm
Width of control channel	30 μm
Width of control valve	100 μm
Minimum spacing between control channels	40 μm
Minimum spacing between external ports	1500 μm

4.4.2.2 Control Layer

In this stage, the placement of the control valves and the control pins $\mathcal{Z}_c$ is decided, and then the valves are connected to the control pins through control channel routing $\mathcal{R}_c$. In Figure 1b, the control layer is shown in red, with the control valves labelled as v_x and the control pins as z_y. Positions of the valves that are used inside a microfluidic component can be obtained directly from the component library. The positions of the valves that need to be placed on the flow channels are inferred from the flow routing information (e.g., valves need to be placed at all flow channel intersections). As explained in Chapter 2, one logical control layer can have two physical layers that can be used for placement and routing (above and below the flow layer) [55]. Contrary to the flow channels, control channels are not allowed to intersect.

After the placement is complete, the next step is to connect the valves to the ports using control channels. The control channels can be routed over/ under any flow channel/ component without forming a valve. The crossing of the control channel over a flow channel forms a valve only if the control channel has a large width (100 μm) [15]. The flow path channel lengths (used to calculate the routing latencies) and any additional routing constraints (imposed because of net intersections in the flow layer) can now be extracted from the layout

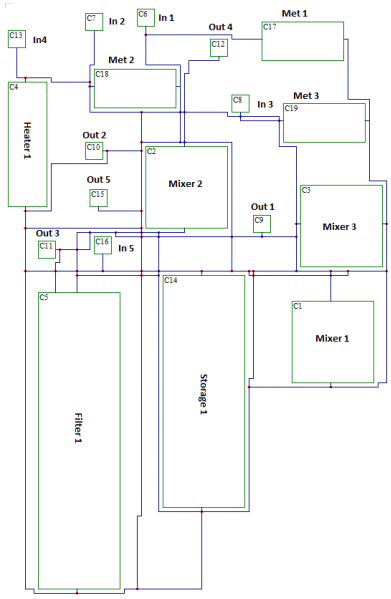


Figure 4.2: Placement and Routing

and captured in the biochip architecture model $\mathcal{A}$. Table 4.3 shows the routing constraints and Table 4.2 shows the list of flow paths for the biochip architecture in Figure 4.1b together with their corresponding routing latencies.

Once the architecture is synthesized and the model is extracted, we can use the approach given in Chapter 3 to map the given application onto the synthesized architecture. Figure 4.3 shows the schedule when the application in Figure 4.1a is scheduled on the architecture in Figure 4.1b. The schedule is represented as a Gantt chart, where, we represent the operations and fluid routing phases as rectangles, with their lengths corresponding to their execution time.

4.5 Synthesis Strategy

The biochip synthesis problem presented in Section 4.3 is NP-complete [84]. Our synthesis strategy is to solve each design task separately, by adapting well-known heuristic algorithms from VLSI domain. The heuristics do not guarantee obtaining the optimal solution. Obtaining the optimal results (in terms of application completion time) is infeasible even for small examples. The following

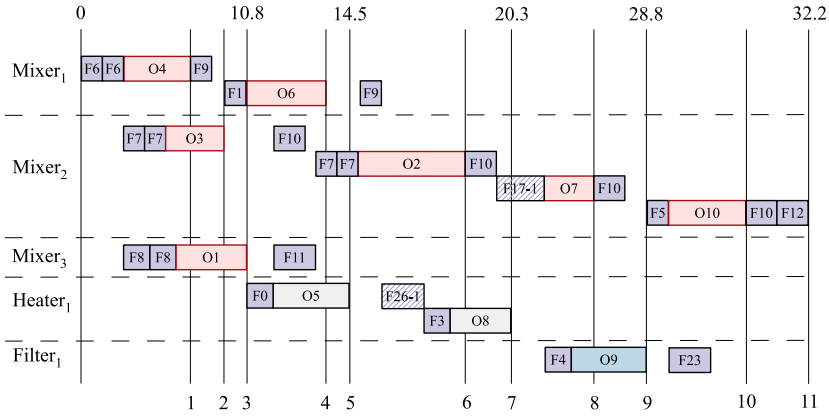


Figure 4.3: Schedule

subsections present the chosen heuristics and describe our strategy using Figure 4 as an illustrative example.

4.5.1 Allocation and Schematic Design

This stage receives the application graph $\mathcal{G}$, component library $\mathcal{L}$ and the resource constraints as input and determines the allocation $\mathcal{U}$ and generates the netlist. All components of the architecture model $\mathcal{A}$ are captured here except the routing latency c.

4.5.1.1 Allocation

The most common approach for allocation in High-Level Synthesis (HLS) [26] is to use resource-constrained List-Scheduling and binding. We start off by topologically sorting the operations of the biochemical application based on their dependency constraints and then prioritizing them using an *urgency criteria* [26]. The urgency of an operation is specified by the length of the longest path from the operation to the end node in the application graph. An operation is considered *ready*, if all of its predecessors have finished execution. All the operations in the application are evaluated, the ready ones are found and are placed in a ready list RL. For example in Figure 4.1a, operations O_1 to O_4 have no predecessors and are thus considered ready, whereas O_6 cannot be executed until O_3 and O_4 are complete. For each ready operation we allocate a compo-

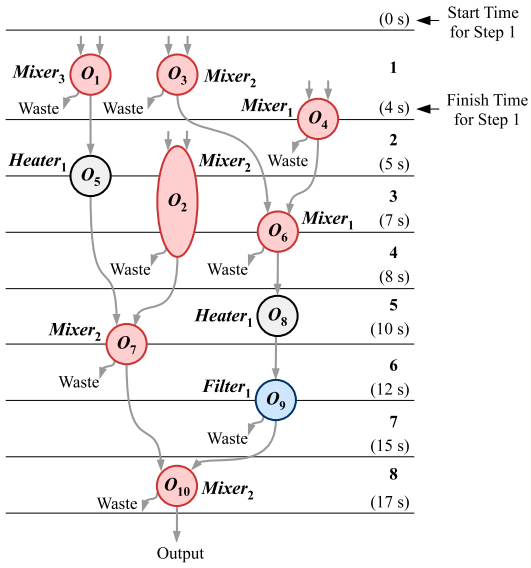


Figure 4.4: Allocation Example for Figure 4.1a

nent of the required type, considering the imposed constraints (see Table 4.1, column 2). The operation is bound greedily to the allocated component and scheduled.

Figure 4.4 shows the allocation schedule for the application in Figure 4.1a. The schedule is divided into 8 schedule steps. The start of an operation marks the start of a schedule step (O_1, O_3, O_4 start at time t = 0 s, thus starting schedule step 1) and an operation completion marks the end of a schedule step (schedule step 1 ends at 4 s as operations O_1 and O_3 finish, and schedule step 2 ends at 5 s when operation O_4 finishes). Unlike the control steps in HLS [26] (where all control steps represent a fixed time duration, a clock cycle), the schedule steps are of varying time lengths.

The binding of each operation is also shown in Figure 4.4, the component name is placed next to the operation (e.g., O_1 is bound to $Mixer_3$). If the number of ready operations exceeds the number of available components, then the least urgent operations (i.e., greedy binding based on the urgency criteria) are deferred, e.g., in Figure 4.4, O_2 is deferred to schedule step 2 as there are only three mixers available for usage in schedule step 1.

As soon as an operation completes, it marks the end of a schedule step. The operations are then re-evaluated to find the new list of ready operations and

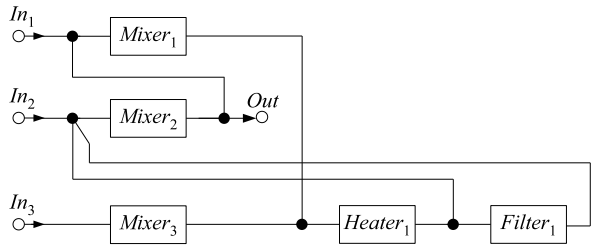


Figure 4.5: Schematic

the process is repeated. Table 4.1 shows the list of allocated components. All imposed resource constraints have been fulfilled. All components have been used in their maximum allowed number except the heater. Only one heater has been allocated considering the requirement, compared to the 2 heater units that were allowed by the user. At this stage, the routing latencies are not yet known. The actual values of the routing latency are generated after the placement and routing is complete. A set of input and output ports is allocated for each component in order to serve as the source and sink point during flow path execution. Metering units are used to create discretized samples of unit volume. The number of metering units allocated depends on the maximum possible external inputs that can be executed in parallel. In the current case, a maximum of three external inputs are taken in parallel in schedule step 1 (Figure 4.4) and thus three metering units have been allocated.

4.5.1.2 Schematic Design

In the next step, we extract the schematic design from the generated binding and scheduling information (analogous to data path generation in HLS [26]). Each schedule step (Figure 4.4) is scanned to find the input source of the utilized components and a corresponding net is placed between the component and the input source. For example, $Heater_1$ in schedule step 5 gets an input from $Mixer_1$ and gives an output to $Filter_1$, therefore it is connected to both components. The extracted component interconnection scheme is shown in Figure 4.5.

Next we connect input and output ports to each component to serve as the source and sink point. Metering units are placed at the fluidic sample input ports. Storage units are needed to store intermediate results of operations [85]. Unlike in HLS (where registers are required at every control step [26]), biochips require storage only under special conditions. Consider an operation O_x bound to a component M_y that finishes execution. If another operation gets bound to M_y in the next schedule step and the successor operation of O_x has not been scheduled

yet, then the output of O_x will need to be moved to the storage. Each storage unit is capable of storing multiple fluid unit samples, depending on the number of storage channels inside the unit. Since the routing latencies are not yet known, it is not possible to accurately assess which components would require the storage unit usage. For now, the storage unit is is connected to all components and the designer specifies the maximum capacity of the unit. Unnecessary connections and extra storage channels are removed after the application mapping step. The final component interconnection configuration is shown in Figure 4.1b.

The flow path set $\mathcal{F}$ and the corresponding routing constraints $\mathcal{K}$ are also generated in this step. The flow path set for the biochip architecture in Figure 4.1b is shown in Table 4.2. The source-sink path for each flow path and the routing constraints are also shown.

4.5.2 Physical Synthesis

This stage takes the allocation $\mathcal{U}$, the netlist, component library $\mathcal{L}$ and the desired layout size as input and performs placement and routing ($\mathcal{Z}_f$, $\mathcal{R}_f$, $\mathcal{Z}_c$, $\mathcal{R}_c$), determining any additional constraints in the set $\mathcal{K}$ and the routing lengths of the flow paths (that are used to calculate the routing latencies c). We use a grid-based approach to perform physical synthesis. The grid size is dictated by the design rules and the component sizes on the grid are calculated accordingly using their dimensions given in the component library. The design rules imposed by the foundry and followed during the physical synthesis are summarized in Table 4.4. The placement and routing phases are divided into two stages:

4.5.2.1 Flow Layer

According to the problem formulation from Section 4.3, the placement and routing should be done such that the application completion time is minimized. However, this would require using the mapping and scheduling (Section 4.5.3) as the cost function, which is too time-consuming. Instead, we use the total channel length and the number of net intersections as the cost function. Minimizing the channel length minimizes the routing latencies, in turn minimizing the application completion time. Similarly, minimizing the number of net intersection minimizes the number of routing constraints, allowing more flow paths to be executed in parallel. This will not lead to the optimal result, but will reduce the application completion time. Furthermore, performing actual routing to compare various placement solutions is impractical as routing is a time-consuming process. Therefore, we perform placement and routing in sepa-

rate steps and use an estimation method (Manhattan distance between two pins is the estimated channel length needed to connect them) to estimate the total channel length in order to judge the quality of the placement solution.

The placement of components such that the total channel length is minimized is an NP-complete problem, for which a number of good heuristic techniques have been developed [69]. Considering the problem at hand, we use Simulated Annealing (SA) (one of the most used methods for cell placement in VLSI [69]) for performing component placement on the chip. Various algorithms have been proposed for routing over the years. We use Hadlock's Algorithm (HA) [69] and extend it for the flow layer routing. HA is suitable for the current problem since it also uses a grid model approach, finds the shortest path between two vertices (if such a path exists) and is faster than the other algorithms in this category [69]. We extend HA to also consider the possibility of net intersections, ensuring a 100% routability. The quality of the solution is judged by the number of net intersections and the total channel length. Since HA is sensitive to the order in which the nets are routed, we iterate on HA, providing it a re-ordered netlist for every iteration in order to achieve a routing solution that minimizes the total channel length and net intersections.

The routing latency corresponding to each flow path is also generated in this step. Routing latencies are calculated by using the routing length of each flow path extracted from the architecture and the flow rate used on the chip. We consider a flow rate of 10 mm/s for all experiments in this chapter. If the flow path length is 10 mm and the flow rate is set at 10 mm/s, then a unit volume of liquid (10 mm length on the channel) traverses this flow path in 2 s, i.e., from the time the tip of the 10 mm unit sample enters the flow path till the time the tail leaves from it. The latency values are required while performing application mapping. Latency values generated for each flow path are shown in Table 4.2.

Physical Synthesis Algorithm.

Figure4.6 shows our algorithm for the physical synthesis of the flow layer. The algorithm takes the allocated component set M, the generated netlist $List$ and the component library $\mathcal{L}$ as an input, and returns the placement and routing information of the flow layer. The objective is to place all the components on the chip and minimize the total channel length in order to reduce the routing latencies, while satisfying the design rules. Figure 4.2 shows the flow layer placement and routing scheme that comes out of our algorithm.

Simulated Annealing [69] (lines $1-17$ in Figure4.6) is used for generating the placement scheme. SA is a metaheuristic, which, starting from a random initial placement of components (line 3), iteratively obtains a better placement scheme by performing *moves* (line 6), i.e., design transformations, (swapping, rotating

FlowPlaceAndRoute(M, $List$, $\mathcal{L}$)

1 *// Phase I: Flow Layer Placement*
2 *Initialize* T
3 $\mathcal{Z}_f^{now}$ = InitialPlacement(M, $\mathcal{L}$)
4 **repeat**
5 **for** i=1 to TL **do**
6 $\mathcal{Z}'_f$ = moves($\mathcal{Z}_f^{now}$)
7 δ = cost($\mathcal{Z}'_f$) − cost($\mathcal{Z}_f^{now}$)
8 **if** $\delta < 0$ **then**
9 $\mathcal{Z}_f^{now} = \mathcal{Z}'_f$
10 **else**
11 **if** $random(0,1) < e^{-\delta/T}$ **then**
12 $\mathcal{Z}_f^{now} = \mathcal{Z}'_f$
13 **end if**
14 **end if**
15 **end for**
16 $T = \alpha \times T$
17 **until** <stop criterion is met>
18 *// Placement returns the best solution* $\mathcal{Z}_f$ *in terms of the cost function*
19 *// Phase II: Flow Channel Routing*
20 $\mathcal{R}_f^{now}$ = RouteFlowLayer($\mathcal{Z}_f$, $List$)
21 c^{now} = cost($\mathcal{R}_f^{now}$)
22 **repeat**
23 $List$ = Re-order($List$)
24 $\mathcal{R}'_f$ = RouteFlowLayer($\mathcal{Z}_f$, $List$)
25 c' = cost($\mathcal{R}'_f$)
26 **if** $c' < c^{now}$ **then**
27 $c^{now} = c'$
28 $\mathcal{R}_f^{now} = \mathcal{R}'_f$
29 **end if**
30 **until** <stop criterion is met>
31 **return** $< \mathcal{Z}_f, \mathcal{R}_f^{now} >$

Figure 4.6: Physical Synthesis Algorithm for the Flow Layer

or randomly changing component location on the chip) to modify the current solution. SA also accepts deteriorations in cost (lines 11−13) to a limited extent in an effort to obtain global optimum, in terms of the cost function used. The placement generated by SA $\mathcal{Z}_f$ is given as input to the Hadlock's Algorithm [69] (lines 20−30) to iteratively generate the routing. A re-ordered netlist is generated (line 23) in every iteration in order to cater for HA's sensitivity to the order in which the nets are routed. The best solution for the placement $\mathcal{Z}_f$ and the routing $\mathcal{R}_f^{now}$ is then returned (line 31).

4.5.2.2 Control Layer

Control layer placement and routing can be done using the same algorithms as described for the flow layer. We aim to target this step in our future research. Since the number of control valves on these chips can be extremely high (commercial chip having more than 25,000 valves [66]) and the number of punch holes that can be made on the chip for connecting the control pins is limited by the design rules [15], each valve cannot be connected to a separate control pin. In the next chapter, we propose an approach for sharing the control pins between valves in order to minimize the number of control pins.

Next, we map the application onto the synthesized architecture using the approach given in Chapter 3. We use the same binding as the one generated during the schematic design (Figure 4.4). As shown in Figure 4.3, the application requires only one storage reservoir. The output of operation O_5 is moved to the storage since the heater needs to be reused for operation O_8 before the successor operation of O_5 (which is operation O_7) is released. The application execution is completed in 32.2 s.

4.6 Experimental Evaluation

We evaluate our proposed approach by synthesizing biochip architectures for three real life assays and a set of four synthetic benchmarks (all given in Appendix B). We implement these applications onto the synthesized chips and determine the application completion time. The algorithm was implemented in C#, running on a Pavilion laptop (HP dv6-2155dx) with Core i3, Dual Processors at 2.13 GHz and 4 GB of RAM.

Table 4.5 shows our experimental results for real-life case studies. Column 1 presents the application and column 2 shows the list of allocated components,

Table 4.5: Real-Life Applications

Appl.	Allocated Units	Chip Area	Net Length	Total Inters.	Total Valves	$\delta_{\mathcal{G}}$
PCR	(3, 3, 3, 0, 0, 0)	250 × 250	198	4	67	19.7 s
IVD	(5, 5, 3, 0, 0, 3)	250 × 250	393	10	101	20 s
CPA	(5, 5, 5, 0, 0, 3)	250 × 250	1360	51	295	72.7 s
EA	(5, 5, 3, 1, 1, 0)	150 × 150	1917	63	311	32.2 s

in the following format: (Input ports, Output ports, Mixers, Heaters, Filters, Detectors). Columns 3−6 present the desired chip area, total length of the flow channels, total number of net intersections and the total number of valves on the chip, respectively. Chip area represents the area given by the user as input. Chip area and total channel lengths are scaled, with a unit length being equal to 150 μm, i.e., a total length of 10 given in Table 4.5 corresponds to 1500 μm. The number of valves are calculated by considering 1 valve for each I/O port, 4 valves for each intersection (switch), 9 valves for each mixer, 6 valves for each metering unit and 2 valves each for all remaining components. The last column presents the completion time $\delta_{\mathcal{G}}$ of the application, in seconds, on the synthesized architecture.

The first real-life assay we use is the PCR (polymerase chain reaction) mixing stage. The architecture details and the corresponding application completion time are shown in row 1 of Table 4.5. Row 2 shows the architecture generated for Multiplexed IVD (in-vitro diagnostics). The third row shows a larger real-life application, a colorimetric protein assay (CPA, 55 operations). It uses a chip equipped with 295 valves to complete its execution in 72.7 s. The architectural details given in row 4 are for the example application (EA) given in Figure 4.1a.

In the second set of experiments we have evaluated our proposed method using a set of four synthetic benchmarks. The benchmark applications are composed of 10, 30, 40 and 50 operations. Table 4.6 shows the details of the synthesized architectures considered and the respective application completion times.

For each application in Table 4.6, two sets of architectures were synthesized. The first row presents results for the architecture synthesized under designer-given constraints (maximum number of components of a certain type is constrained), whereas, the second row presents the results of an unconstrained architecture,

Table 4.6: Synthetic Benchmarks

$\lvert\mathcal{O}\rvert$	**Allocated Units**	**Chip Area**	**Net Length**	**Total Inters.**	**Total Valves**	$\delta_{\mathcal{G}}$
10	(2, 2, 1, 1, 1, 1)	150 × 150	1813	45	211	39.9 s
	(5, 5, 2, 1, 1, 1)	150 × 150	1926	68	324	35.7 s
30	(6, 6, 3, 2, 2, 1)	250 × 250	3575	122	573	64.5 s
	(15, 18, 6, 4, 3, 1)	350 × 350	5243	124	665	46.1 s
40	(8, 8, 4, 3, 1, 2)	350 × 350	4799	151	716	69.8 s
	(18, 20, 7, 5, 2, 3)	350 × 350	7452	171	889	59.5 s
50	(10, 10, 5, 2, 2, 2)	350 × 350	6522	177	839	81.25 s
	(21, 24, 10, 4, 3, 3)	400 × 400	9366	213	1109	60.1 s

i.e., no constraints were placed on the number of components to be used. Allocation step for the unconstrained architecture case can be considered similar to ASAP Scheduling [56]. For all applications, the unconstrained architecture produces a completion time that is smaller than that of the constrained architecture. All experiments presented in this section took between 3 to 30 minutes to complete, depending on the complexity of the application. All benchmarks and architecture details can be found here [14].

4.7 Summary

In this chapter we have presented a top-down architectural synthesis approach for flow-based microfluidic biochips. The proposed approach synthesizes a biochip architecture for a given biochemical application, such that the application completion time is minimized. The synthesis process involves component allocation, design schematic generation, and the physical synthesis (placement and routing) of the chip. The approach has been evaluated by synthesizing biochip architectures for three real-life assays and a set of synthetic benchmarks. The proposed approach is expected to facilitate programmability and automation in the microfluidics domain, reducing human effort and minimizing the design cycle time.

Once the biochip architecture has been synthesized and the application mapping step has been completed, the output of the application mapping block can be used to synthesize input for the biochip controller. This will enable automatic execution of the given biochemical application on the synthesized biochip architecture. The following chapter discusses this topic in detail.

Chapter 5

Control Synthesis

In this chapter we focus on the "Control Synthesis" block of our proposed design methodology (see Figure 1.6). We start off by presenting the related work and describing our contributions. The tasks involved in control synthesis are explained in Section 5.3 and our solution strategy is presented in Section 5.4. We experimentally evaluate our strategy in Section 5.5.

5.1 Related Work

There are two steps of control synthesis: (i) *control logic generation* and (ii) *control pin count minimization*. Control logic generation means determining which valves need to be opened or closed, in what sequence and for how long, in order to execute the application on the chip [65]. Currently, this is done manually by the designers [81]. Generating control logic manually is inefficient and error-prone (similar to exposing the gate-level details in microelectronics). Moreover, as the chips become more complex and the assays more concurrent, the manual process will not scale.

The second step is the control pin count minimization. As discussed in Section 1.3.1, a valve is connected to a pressure source through a control pin (e.g.,

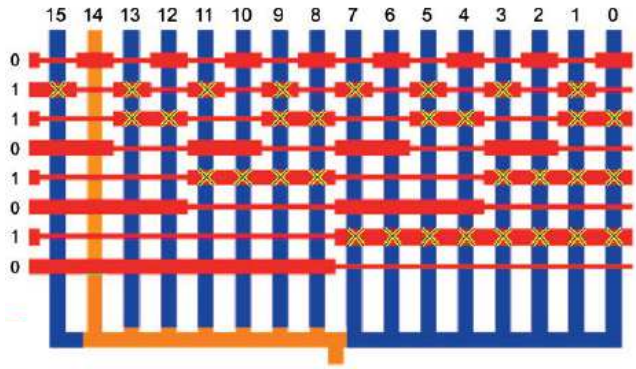


Figure 5.1: Microfluidic Multiplexer [55]

in Figure 1.3a, valve v_a is connected to the pressure source through control pin z_1). A valve is pressurized (closed valve) or de-pressurized (open valve) by the pressure source connected to the control pin. Having a separate control pin for every valve directly translates to a large pin-count for the chip, resulting in (i) high consumption of the chip area, and (ii) requirement of large, expensive and complex macro-assembly around the chip (each valve requiring its own pressure source). High pin-count is a bottleneck to the scalability of these chips and therefore needs to be minimized.

Microfluidic multiplexers [55] have previously been used to minimize the control pin count. Consider that there are N flow channels on the chip requiring N valves to close them. If every valve gets a separate pin and therefore a separate pressure source, N pressure sources will be required. However, if it is known that only one of the N flow channels needs to be open at a time (all others need to be closed), then a microfluidic multiplexer can be used. Such a multiplexer will require only $2log_2$N control pins for controlling N such flow channels [21], e.g., 8 pins will be sufficient to control 16 such flow channels as shown in Figure 5.1. Valves are vreated only where a wide control channel (red) intersects with a flow channel (blue). Narrow intersections do not create valves. In Figure 5.1, control channels with status 1 have the associated valves closed and the ones with status 0 have the associated valves open. For the case shown, the valves are activated such that flow channel 14 is open and all others are closed.

In order to perform the multiplexer-based optimization, the user is asked to manually specify which flows on the chip will be used and if they will be used in parallel or not [21]. Once the flows have been specified, then the multiplexer is used for sharing the control pins between flow channels that satisfy the above criteria. Manual flow specification requires the user to have complete under-

standing of the chip as well as the application requirements. The manual input provided by the user can also easily result in under or over-constrained specifications, resulting in inefficient minimization. Moreover, the multiplexers-based minimization is only applicable if the criteria of "only one out of N flow channels needs to be open at a time" is fulfilled. The control pin count minimization problem has been reduced to the graph-coloring problem (which is NP-hard) [21] but no solution or experimental results have been presented.

5.2 Contribution

We propose a top-down control synthesis framework for implementing biochemical applications on flow-based biochips. Given a biochip architecture and the mapping implementation of a biochemical application on the biochip architecture, control synthesis consists of the following two steps: (i) *control logic generation* and, (ii) *control pin count minimization*. We utilize the output of control logic generation step and perform the minimization by sharing the control pins between multiple valves, provided that those valves operate in unison throughout the entire application execution. Pin count minimization using multiplexers and our proposed approach can be performed one after the other in order to reach a good solution. Here, we do not perform multiplexing but consider that this may already have been done in the given biochip architecture.

Considering the computational complexity of the problem, we propose a Tabu Search [48] based optimization in order to minimize the pin count. Our framework does not require manual flow specifications from the user, decoupling the application design from control synthesis. To the best of our knowledge, this is the first time an approach for the control logic generation for the mVLSI biochips is being presented.

5.3 Biochip Control Synthesis

The following subsections explain the tasks involved in the biochip control synthesis. Our proposed solution is discussed in Section 5.4.

We use the biochip architecture given in Figure 5.2 and the application given in Figure 5.3b as an illustrative example. The biochip has 4 mixers, 2 heaters, 2 filters and 2 detectors. Table 5.1 shows the list of flow paths, their routing latencies and the routing constraints for the architecture in Figure 5.2. Each

Table 5.1: Biochip Flow Path Set ($\mathcal{F}$), Control Layer Model and Routing Constraints ($\mathcal{K}$)

Flow Paths	Control Layer Model			Routing Constraints
	Flow Path	Closed Valves	Open Valves	
$F_{0-x} = (In_1, Mixer_1)$, 2 s	F_{0-1}	4, 7, 67	1, 2, 3, 5, 6, 8, 9, 10, 68	$K_{3-x} : F_{10-x}$
$F_{1-x} = (Mixer_1, Filter_1)$, 2 s	F_{0-2}	3, 5, 67	1, 2, 4, 6, 7, 68, 8, 9, 10	$K_{8-x} : F_{12-x}$
$F_2 = (Filter_1, Heater_2)$, 2 s	...			$K_{10-x} : F_{3-x}$
...	F_{6-1}	30, 33, 71	27, 28, 29, 34, 35, 36, 31, 32, 72	$K_{12-x} : F_{8-x}$
$F_{6-x} = (In_2, Mixer_2)$, 2 s	F_{6-2}	29, 31, 71	27, 28, 30, 33, 32, 72, 34, 35, 36	
...	...			
$F_{11} = (In_3, Heater_1)$, 2 s	F_{11}	64	42, 43, 44, 63	
$F_{12-x} = (Heater_1, S_1, Mixer_3)$, 3 s	...			

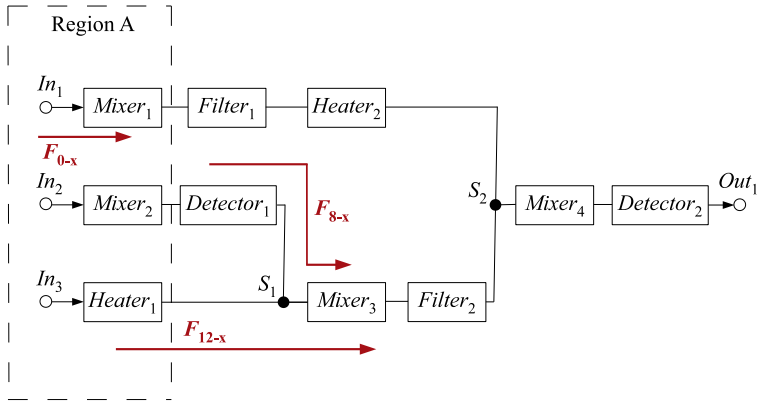


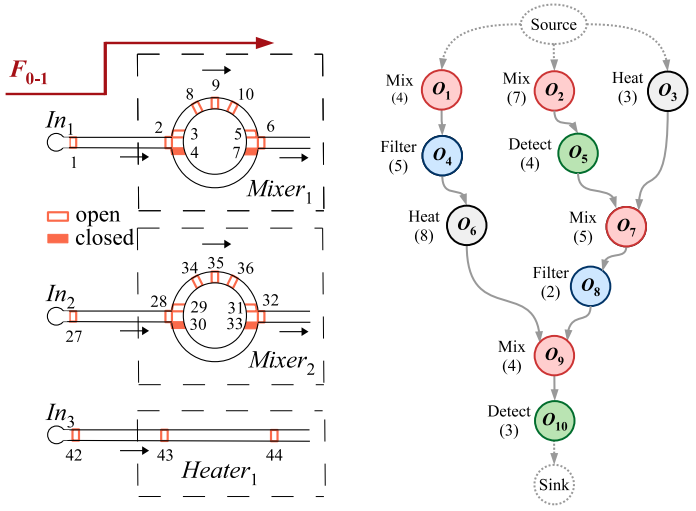
Figure 5.2: Biochip Architecture Example

flow path has an associated control layer model (given in Table 5.1) that contains the details required for its utilization, i.e., the switch and the pump activation details. Figure 5.3a shows the schematic view for the area marked as *Region A* in Figure 5.2. For the flow path F_{0-1} (1 for referring to the upper half of the mixer) that goes from In_1 to $Mixer_1$ (as shown in Figure 5.3a), the valve set {4, 7} needs to be closed and the valve set {1, 2, 3, 8, 9, 10, 5, 6} needs to be opened. The same is given as the control layer model of F_{0-1} in Table 5.1. A pumping action then moves the fluid from In_1 to the $Mixer_1$ upper half. For the biochip architecture in Figure 5.2, we consider that all the flow paths have their own dedicated off-chip pumps. The pumps can, however, be easily shared between flow paths. In that case, the flow paths that share a pump will become mutually exclusive and will be listed as such under the routing constraints.

The components in the architecture also have control layer models. For example, $Mixer_1$ in Figure 5.3a is a pneumatic mixer (introduced in Section 2.1.1) implemented using nine microfluidic valves, numbered 2 to 10. The valve set {8, 9, 10} acts as an on-chip pump. The valve set {2, 3, 4} and the valve set {5, 6, 7} act as switches. The two switches facilitate the inputs and outputs, and the pump performs mixing.

The mixer has five operational phases. The first two phases represent the input of two samples for mixing (filling the upper and lower half of the mixer), followed by the mixing phase (**Mix**). The mixed sample is then transported out of the mixer in the last two phases, emptying the two halves.

In order to perform mixing (once both halves are filled), the mixer input and output valves {2, 6} are closed while valves {3, 4, 5, 7} (see Figure 5.3a) are



(a) Region A in Figure 5.2 (b) Application Graph

Figure 5.3: Schematic View and Application Example

opened and the mixing operation is initiated. Valve set {8, 9, 10} acts as a peristaltic pump. Closing valve 8 inserts some pressure on the fluid inside the mixer, closing valve 9 creates further pressure, then as valve 10 is closed valve 8 is opened again. This forces the liquid to rotate clockwise in the mixer. The valves are closed and opened in a sequence such that the liquid rotates at a certain speed accomplishing the mixing operation. The control layer of the mixing phase is a part of the component model. Table 5.2 shows the control layer model for all components in the biochip in Figure 5.2. The control layer details are only for the functional phase of the component. For example for $Mixer_1$, the control details are for the **Mix** operation for which the valve set {2, 6} is closed, {3, 4, 5, 7} is opened and {8, 9, 10} is in the mixing state (opening and closing in a predefined sequence). Valve sequences for the heaters, filters and detectors are also given. In addition to these valve activations, the relevant component also needs to be activated, e.g., optical sensor present in the detector needs to start operation as soon as the associated valves have been activated.

The input and output phases of the components are modeled using flow paths (and their associated control layers) in the architecture model. For example, the input to $Mixer_1$ from In_1 in Figure 5.3a is modeled by the flow path F_{0-x} (see Table 5.1) and its activation is done using the associated control layer. Note that the mix valves (e.g., valve set {8, 9, 10} for $Mixer_1$) need to stay open for the input and output phases of the mixer. Mix valves are active only when the

Table 5.2: Component Control Layer Model for Figure 5.2

Component	Open Valves	Closed Valves	Mixing Valves
$Mixer_1$	3, 4, 5, 7	2, 6	8, 9, 10
$Mixer_2$	29, 30, 31, 33	28, 32	34, 35, 36
$Mixer_3$	49, 50, 51, 53	48, 52	54, 55, 56
$Mixer_4$	19, 20, 21, 23	18, 22	37, 38, 39
$Heater_1$	-	43, 44	-
$Heater_2$	-	13, 14	-
$Filter_1$	-	11, 12	-
$Filter_2$	-	57, 58	-
$Detector_1$	-	40, 41	-
$Detector_2$	-	24, 25	-

mixing is intended and need to be kept open after the mixing is done, until the desired mixed fluid has been taken out emptying the mixer.

5.3.1 Control Logic Generation

Generating the control logic η means deciding which valves to close/ open, in what sequence, at what time and for how long, in order to implement a biochemical application $\mathcal{G}$ on the chip architecture $\mathcal{A}$. A mapping $\psi =< \mathcal{B}, \mathcal{X} >$ (binding and scheduling information) of the application on the biochip architecture is given as input. Figure 5.4 shows the schedule $\mathcal{X}$ after mapping the application in Figure 5.3b on the biochip in Figure 5.2. The schedule is depicted as a Gantt chart, where we represent the operations and fluid routing phases as rectangles, with the lengths corresponding to their execution times. The start time and the end time for the flow paths and operations in the schedule are called *time steps*, i.e., 0 s and 2 s in Figure 5.4 are considered time steps since they mark the start/ end time of the flow paths, whereas, 1 s is not a time step.

The control logic η is represented in a tabular form and contains the activation status of all valves on the chip, for all time steps of the schedule. The control logic presented in Table 5.3 gives the activation status of the valves shown in Figure 5.3a for the schedule (given in Figure 5.4) duration 0 to 8 s.

Each row in the Table 5.3 represents the activation status of a valve. First column contains the valve number and the remaining columns represent the activation status of the valve for the time steps present in the schedule. For example, the first row in Table 5.3 represents the activation status of valve 1. A

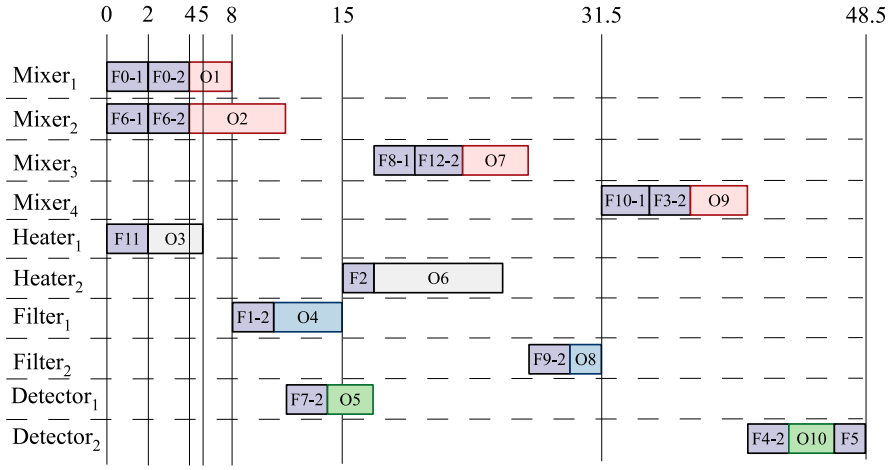


Figure 5.4: Example Schedule

0 as activation status represents an open valve, 1 a closed valve and X represents a don't-care, i.e., the valve may be opened or closed without having any influence on the application execution. For example, valve 1 (row 1 in Table 5.3), is opened at time 0 s, stays open at 2 s and then its status changes to a don't-care at 4 s. This is because from 0 to 4 s, F_{0-1} and F_{0-2} are executed, as shown in the schedule in Figure 5.4, filling the upper and lower halves of $Mixer_1$ (see Table 5.1). At 4 s, both fluid samples are inside the mixer, therefore valve 2 closes in order to start the mixing operation (valve 2 status changes to 1 at 4 s in Table 5.3). Once valve 2 is closed, the status of valve 1 switches to a don't-care. This is because valve 1 and valve 2 are placed in series on the flow channel (see Figure 5.3a) and once valve 2 is closed, opening or closing of valve 1 has no impact on the application execution. The mix valves (e.g., valve 8, 9, 10 in $Mixer_1$) act as a pump in order to achieve mixing [55]. This pumping is also included in η and for simplicity, it is shown as "Mix" in Table 5.3. The mix valves are opened and closed at a certain frequency in order to achieve mixing, and this opening and closing continues even between time steps.

5.3.2 Pin Count Minimization

The biochip architecture may contain some valves that are never closed during the application execution. These valves are redundant and can be removed reducing the pin count, e.g., valve 1 in Table 5.3 is never closed and is therefore redundant. Connecting each valve to a separate control pin results in too many

Table 5.3: Control Logic (η) Table - For Valves in Figure 5.3a

Valve	Time Steps (s)						Color
No.	0	2	4	5	8	...	
1	0	0	X	X	0	...	-
2	0	0	1	1	0	...	Color - 0
3	0	1	0	0	1	...	Color - 11
4	1	0	0	0	0	...	Color - 1
5	0	1	0	0	1	...	Color - 11
6	0	0	1	1	0	...	Color - 0
7	1	0	0	0	0	...	Color - 1
8	0	0	Mix	Mix	0	...	Color - 14
9	0	0	Mix	Mix	0	...	Color - 8
10	0	0	Mix	Mix	0	...	Color - 2
...	...	...	...	...	...	...	...
27	0	0	X	X	X	...	-
28	0	0	1	1	1	...	Color - 3
29	0	1	0	0	0	...	Color - 6
30	1	0	0	0	0	...	Color - 1
31	0	1	0	0	0	...	Color - 6
32	0	0	1	1	1	...	Color - 3
33	1	0	0	0	0	...	Color - 1
34	0	0	Mix	Mix	Mix	...	Color - 9
35	0	0	Mix	Mix	Mix	...	Color - 4
36	0	0	Mix	Mix	Mix	...	Color - 7
...	...	...	...	...	...	...	...
42	0	X	X	X	X	...	-
43	0	1	1	X	X	...	Color - 13
44	0	1	1	X	X	...	Color - 13
...	...	...	...	...	...	...	...
89	X	X	X	X	X	...	Color - 4

pin-outs from the chip limiting the chip scalability. In order to minimize the pin count, a strategy is needed in order to share the control pins between different valves that perform in unison with each other throughout the application execution schedule. For example in Table 5.3, valve 2 and valve 6 have identical activation sequence in all time steps and therefore, can share the same control pin. Similarly, valve 1 and 2 also have the same sequence (X for valve 1 at time steps 4 and 5 means that valve 1 can be switched to 1 or 0 without affecting the application execution) and can share the control pins.

5.3.3 Problem Formulation

Given (1) a biochip architecture modeled as a topology graph $\mathcal{A}$, and (2) a mapping $\psi = < \mathcal{B}, \mathcal{X} >$ (binding and scheduling information) of an application $\mathcal{G}$ on the biochip architecture $\mathcal{A}$, we are interested in performing the control synthesis for the given chip. Control synthesis consists of the following steps: (1) generating the control logic η needed to execute the application on the chip, and (2) minimizing the control pin count of the chip.

5.4 Synthesis Strategy

Figure 5.5 shows our algorithm which is explained in the following subsections.

5.4.1 Control Logic Generation

Control logic η is generated by fetching the control layer model of the biochip flow paths $\mathcal{F}$ and components M (part of the biochip architecture model $\mathcal{A}$), and utilizing them to translate the mapping ψ into the valve activation sequence (line 2 in Figure 5.5). At every time step of the schedule $\mathcal{X}$, we look at the active flow paths and operations, fetch the associated control layer models and populate the table representing the control logic. The valves that need to be opened are given a status 0, the ones that need to be closed 1 and to the set of valves that are mixing the status "Mix" is allotted. All other valves are set as X (don't-care) for this time step and then the algorithm moves on to the next time step. For example at time step 2, operation O_3 and flow paths F_{0-2}, F_{6-2} are active, as shown in the schedule in Figure 5.4. Operation O_3 is bound to $Heater_1$, so we fetch the control layer model for $Heater_1$ from Table 5.2 according to which valves 43, 44 should be closed. The status for these valves is therefore set to 1 at time step 2 in the control logic (Table 5.3). Similarly the control layer models for the flow paths F_{0-2} and F_{6-2} are fetched from Table 5.1 and the valves involved are set to either 1 or 0, depending on whether they needed to be closed or opened. All other valves (except the mix valves) are set to the status X for time step 2.

The mix valves are assigned a don't care status X only when either both halves of the mixer are empty, or when the mixed fluid in only one half of the mixer was required for the application and that half has been emptied. When mix valves (e.g., {8, 9, 10} for $Mixer_1$) are set to X, the input and output valves of the respective mixer ({2, 6} for $Mixer_1$) need to be closed. This ensures that

ControlSynthesis($\mathcal{A}$, ψ, k, maxIter)

```
1  // Generate control logic
2  η = GenCtrlLogic(F, M, ψ)
3  // Algorithm for pin count minimization
4  Gc = GenGraph(η)
5  k = |Gc|, kIterate = 1
6  while f(s) > 0 and kIterate = 1 do
7     // Generate initial solution
8     s = Random(Gc)
9     Initialize TL to φ, nrIter = 0, s* = s
10    while f(s) > 0 and nrIter < maxIter do
11       BestMove(s') = GenNbrSelectMove(s, TL)
12       Update TL
13       if f(s*) > f(s') then
14          s* = s', nrIter = 0
15       else
16          nrIter++
17       end if
18       PerformMove {s = s'}
19    end while
20    (k, kIterate) = BSearch(f(s), k)
21 end while
22 return  < η, G_C, s*, k>
```

Figure 5.5: Synthesis Algorithm

if these mix valves (e.g., {8, 9, 10} of $Mixer_1$) share control pins with other mix valves (e.g., {34, 35, 36} for $Mixer_2$) and a pumping action is performed because of this, the pumping affect is contained inside the mixer and does not affect the rest of the chip operation.

5.4.2 Pin Count Minimization

The pin count minimization problem has previously been reduced to a graph coloring problem (GCP) [21]. In GCP, the nodes in the graph need to be colored using minimum number of colors, in such a way that no two adjacent nodes have the same color. Finding the exact chromatic number (the minimum number of colors that can be used to color the graph nodes) is an NP-hard problem [21]. Finding out if the graph can be covered with k colors is an NP-complete problem [48]. The problem has previously been reduced to a GCP, however, no solution or experimental results have been proposed.

5.4.2.1 Graph Generation

Before we generate the graph, we remove redundant valves, if any, from the biochip architecture. Redundant valves are the ones that are never closed during the entire application execution, e.g., valve 1, 27 and 42 in Table 5.3 are redundant valves as their status is never set to 1. These valves can be removed from the biochip as their presence has no effect on the application execution.

Next, we create the graph $\mathcal{G}_C(\mathcal{V}_C, \mathcal{E}_C)$ (line 4 of Figure 5.5) by considering each valve in Table 5.3 as a separate node Vc in the graph (redundant valves are not considered). An edge Ec is made between two nodes if a time step exists in the schedule for which the valves (represented by the nodes) have a different activation status. For example, the nodes representing valve 2 and valve 6 will not have an edge between them as they operate in unison throughout the schedule as shown in Table 5.3, but an edge will be made between valve 2 and 3 since their activation status vary at time step 2 (valve 2 is open and valve 3 is closed). The graph is complete once all edges have been drawn. The graph for Table 5.3 has 83 nodes (total valves were initially 89, 6 were found to be redundant and were removed) and 1312 edges.

5.4.2.2 Pin Count Minimization Algorithm

The problem for pin count minimization is now represented in the form of a classical graph coloring problem (GCP). Once the colors have been assigned, the nodes that have the same color will share the same control pin.

GCP has been studied extensively and different approaches have been proposed to find a good quality solution. The simplest approach is the Greedy method [48] which takes the ordering of the nodes as input and colors them with the smallest color number, while satisfying the constraint that no two adjacent nodes should have the same color. Greedy often performs poorly in practice since a good node ordering is difficult to decide. DSATUR is another commonly utilized approach that uses a heuristic to dynamically change the ordering of the nodes and then uses the Greedy method for coloring [48]. Branch and bound algorithms [48] and semi-definite programming solutions for approximate graph coloring [41] have also been proposed. Considering the complexity of the problem, different metaheuristic techniques have also been used extensively for finding good graph coloring solutions, especially when there is a large number of nodes [48]. We use a Tabu Search-based optimization scheme in order to perform the pin count minimization.

Tabu Search (TS) is a metaheuristic, based on the neighbourhood search, which uses design transformations (moves) applied to the current solution in order to generate a set of neighbouring solutions that can be further explored by the algorithm [34, 37]. Our algorithm targets a k-GCP, i.e., finding a graph coloring using k number of colors and then iterates to find the smallest value of k.

Given the graph $\mathcal{G}_C(\mathcal{V}_C, \mathcal{E}_C)$ to be colored, the target of the algorithm is to partition the nodes into a fixed k number of subsets, such that no two adjacent nodes belong to the same subset. We start with a random solution s for the k-coloring (line 8 in Figure 5.5), which (typically) contains a high number of edges in a conflict, i.e., the nodes connected by these edges have the same color. Using this as an initial solution the algorithm iterates to explore the solution space S, which is the set of all possible k-colorings of the graph.

In order to efficiently perform the search, TS uses memory to record moves that are not allowed at the present iteration (called tabu list (TL)). The aim is to exclude moves that would cycle the search back to the local optima that has already been evaluated. A move remains tabu for a certain number of iterations (called the size of tabu list). Our algorithm starts with an empty TL and the best known solution $s*$ is initialized with the initial solution s (line 9).

Next, the algorithm generates neighbourhoods for the current solution s and picks the best one from these based on a certain objective function (line 11). A neighbour s' is generated by selecting a random node x that is adjacent to an edge in conflict. Assuming $x \in V_{Ci}$, we choose a random color $j \neq i$ and replace i with j to obtain s'. The objective function $f(s)$ is given as:

$$f(s) = \sum_{i=1}^{k} |E_{Ci}| \tag{5.1}$$

(it measures the number of edges in conflict), where $s \in S$ is the solution under evaluation, and E_{Ci} is the set of edges in conflict, having color i. The objective of the algorithm is to determine a k-coloring such that $f(s) = 0$. The best move is the one that is not tabu and has the lowest $f(s)$ value.

Our algorithm generates the tabu list as follows: whenever a node x is moved from V_{Ci} to V_{Cj}, the pair (x, i) becomes tabu, i.e., node x cannot be returned to V_{Ci} for a certain number of iterations. However, in order to not prohibit attractive moves, our algorithm uses the aspiration criteria of allowing the tabu moves if they result in a solution s' that is better than the currently known best solution $s*$. The best move is selected and the tabu list is updated (line 11, 12).

The algorithm continues searching until it finds a solution for which $f(s) = 0$ or until it reaches the maximum number of iterations without an improvement

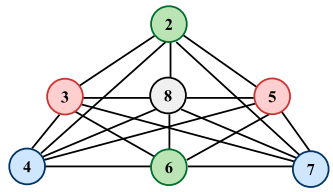


Figure 5.6: Colored Graph

in the solution (number of iterations nrIter is reset and incremented in line 14 and 16 respectively). Our algorithm uses binary search (half-interval search) to select new value of k, as it iterates to find the minimum value of k for which $f(s) = 0$ (line 20). When the minimum k is found, the algorithm stops and the control logic η and the pin sharing information ($\mathcal{G}c, s*$, k) is returned (line 22). The graph generated for Table 5.3 has 83 nodes, i.e., 83 valves requiring 83 control pins and therefore 83 off-chip pressure sources. After performing the pin count minimization, only 15 control pins were found to be needed to control these 83 valves and to execute the control logic given in Table 5.3. Last column in Table 5.3 shows the colors assigned to the valves (valves with the same color share the control pin). A small section of the colored graph is shown in Figure 5.6 representing the pin sharing, e.g., valve 2 and valve 6 do not have an edge between them, hence they share the same color (and therefore the same control pin). Complete graph is shown in Figure 5.7.

5.5 Experimental Evaluation

We evaluate our proposed approach by synthesizing real-life assays as well as synthetic benchmarks. The algorithm was implemented in C++, running on Lenovo T400s ThinkPad with Core 2 Duo 2.53 GHz Processors and 4 GB RAM.

Table 5.4 shows the results. Column 1 shows the application name (for a real-life assay) or the number of operations in the application (in case of a synthetic benchmark). The second column gives details of the biochip architecture in the following format: (Mixers, Heaters, Filters, Detectors). The third column shows the number of control pins originally needed and column 4 shows the number of control pins needed once the redundant valves have been removed (the term NR in the table means Non-Redundant). Column 5 gives the final pin count after the minimization has been performed.

The first real-life assay is PCR (polymerase chain reaction) mixing stage that has 9 mix operations and is used in DNA amplification [14]. Next, real-life ass-

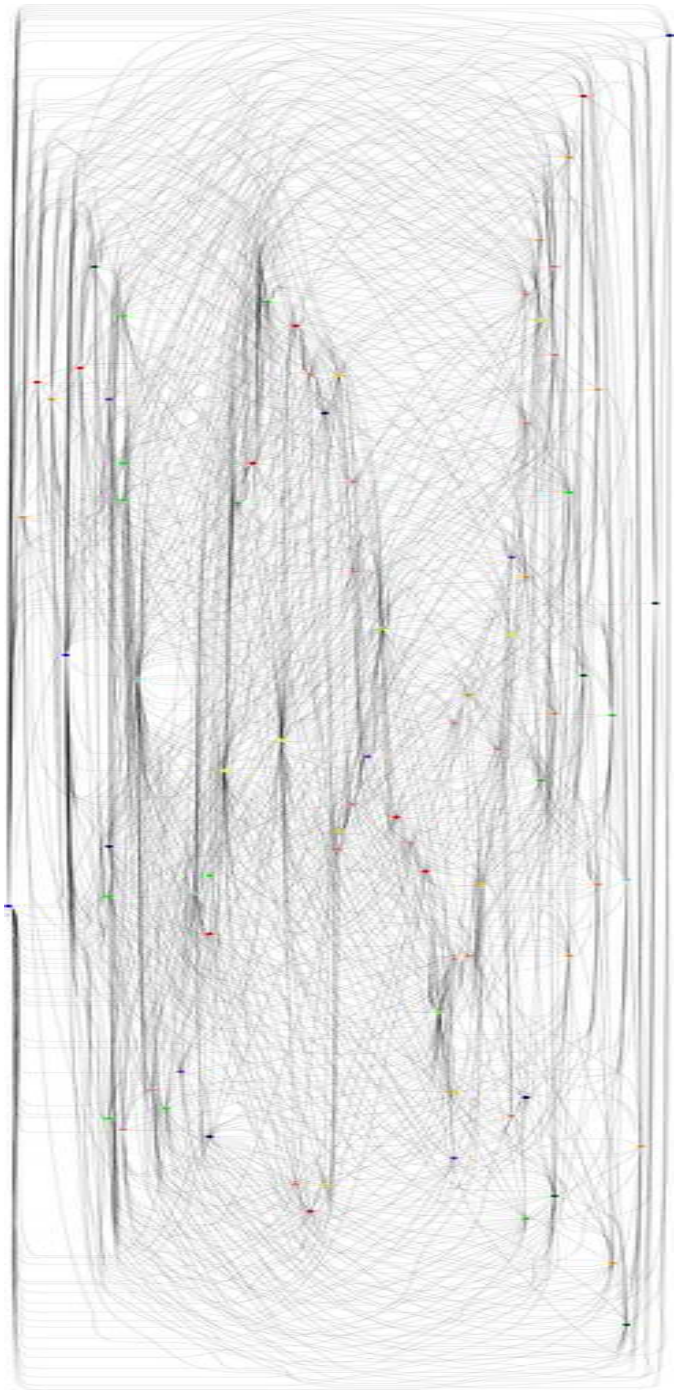


Figure 5.7: Complete Graph for Table 5.3

Table 5.4: Experimental Results

Appl./ Nodes	Allocated Units	Total Pins	NR Pins	Optimized Pins
PCR	(4, 0, 0, 0)	73	52	12
IVD	(6, 0, 0, 6)	65	65	12
10	(4, 2, 2, 2)	87	83	15
20	(12, 4, 3, 1)	164	145	38
30	(17, 6, 4, 3)	212	201	47
40	(21, 9, 6, 4)	269	197	74
50	(26, 12, 7, 5)	329	240	61

ay Multiplexed IVD (in-vitro diagnostics) has a total of 12 operations and is used to test fluid samples from the human body. The synthetic benchmark applications are composed of 10 up to 50 operations. In all cases, the pin count is reduced by more than 70%, e.g., for the 10 node application it goes down from 87 to 15, an 82% reduction in pin count. The parameter values were determined by tuning. Size of TL was set to 7 and the number of neighbours/ sample to 15. The algorithm execution time depends on the chosen value of k and the size of the problem. On average, for each value of k, the algorithm iterates for 2 to 4 minutes. All architectures and benchmarks can be found here [14].

5.6 Summary

In this chapter, a top-down control synthesis framework for implementing the biochemical applications on the flow-based biochips has been proposed. Our algorithm generates the control logic needed to execute the application and uses a Tabu Search-based optimization in order to minimize the control pin count. The minimization is targeted at efficiently utilizing the chip area and reducing the macro-assembly around the chip. The framework has been evaluated using real-life applications as well as a set of synthetic benchmarks. The proposed approach is expected to facilitate programmability and automation, and enhance scalability of the flow-based biochips.

Chapter 6

Experimental Throughput Maximization for Cell Culture Biochips

The term "cell culturing" refers to the process of growing cells outside their natural environment (in vitro). This allows the study of complex cellular mechanisms without requiring the usage of laboratory animals. Cell culturing provides biological insights into cells and tissues bringing exciting opportunities for stem cell research, drug discovery (e.g., for cancer) and a wide range of other applications in the biomedical and pharmaceutical industry.

In cell culturing, an *experiment* is defined as the exposure of a cell colony to a sequence of fluidic compounds and monitoring its response. For example, cancer cells can be exposed to a sequence of fluidic compounds (possible drugs) in order to monitor which compounds (or which sequence of compounds) has the most desired effect. Exposing cells to all possible sequences of the fluidic compounds is called a *full factorial experiment* and exposing them to a fraction of all sequences is called a *fractionally factorial experiment* [63].

Drug discovery is one of the foremost applications of cell culturing. The first step to drug discovery is to screen millions of compounds for effects on one target (e.g., a cell colony). Since there are millions of compounds to screen for,

only a fractional factorial design is utilized. Once the lead compounds have been identified, a full factorial design is initiated. Full factorial experiments are expensive in terms of time and cost and hence need to be performed using high throughput techniques to conserve resources [75]. Having high throughput means performing as many experiments as possible using the available cell colonies and fluidic compounds, within the limited amount of time.

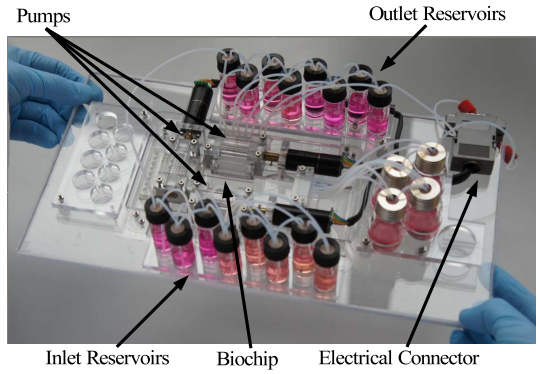
Biologists have traditionally used cell culture platforms equipped with microwell plates, where the cells are cultured inside the wells. Seeding, growing, feeding and analysing the cell cultures inside the wells manually, requires many hours of repetitive and painstaking work. Therefore, cell culture platforms have been automated using robotics. Robotics-based automated cell culture systems are being used today to increase the throughput by using their ability to manage a large number of experiments, e.g., a cell culture system featuring a CRS 465 robotic arm can handle up to 504 microwell plates [42]. Flow-based biochips offer a promising alternative to robotic microwell cell culture systems [27]. Biochips can mimic both the complex biochemistries and the geometry of environments found in organisms. At the same time, transport of fluids and soluble compounds is regulated through the microfluidic channels creating new opportunities for the spatial and temporal control of cell growth and stimuli [28].

Our focus in this chapter is on modeling a cell culture biochip architecture and optimizing how a biochemical application is executed on these biochips, such that the experimental throughput is maximized. The architecture modeling is presented in Section 6.1.1 and the optimization problem in Section 6.1.2.

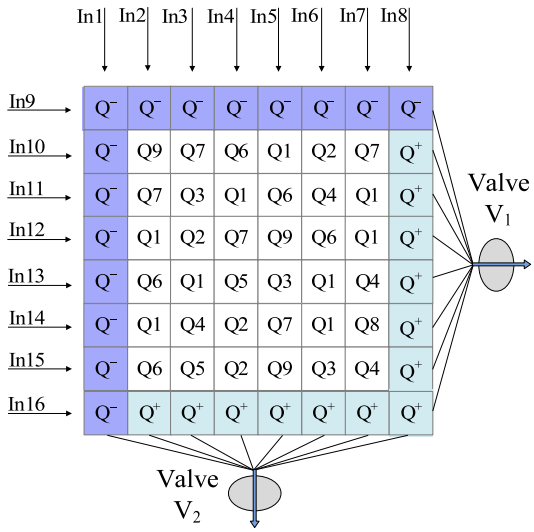
Several microfluidic cell culture platforms have been proposed. A large number of cell culture platforms contain biochips that have been fabricated using PMMA (polymethyl methacrylate) as the substrate. PMMA is more suitable than PDMS in culturing certain cell types. We start by considering PMMA-based cell culture biochips and later explain the mVLSI-based ones. The techniques proposed in this chapter are applicable to both PMMA and mVLSI biochips.

Consider the platform shown in Figure 6.1 [76]. It has a PMMA biochip on which the cells are cultured, inlet reservoirs containing compounds to which the cells are exposed, pumps that are used to push the compounds into the chip and outlet reservoirs that collect the compounds when they come out after passing over the cells. The system is programmable, providing simultaneous software control of the pumps and the microscope for automated image analysis.

The cell culture biochip architecture, shown in Figure 6.1b, provides the opportunity for performing these high throughput experiments. The architecture shown in the figure can hold 64 cell colonies and therefore 64 experiments simultaneously (8×8 matrix, see Section 6.1.1 for details). All the experiments being



(a) Biochip Platform [86]



(b) Laminar Flow Biochip

Figure 6.1: Biochip Platform and Architecture

carried out (64 in this case) in the chip are collectively considered as one *experimental stage*. An experiment is considered unique if it is not repeated anywhere on the chip. The biochip throughput depends on the number of unique experiments carried out on the chip. Note that several experimental stages might be necessary to achieve a certain target objective. A new chip (or the same chip might be washed and reused) has to be used for the next experimental stage.

An experimental stage has two steps. (1) In the first step the cell colonies are inserted and placed on the chip. The placement is fixed for an entire experi-

mental stage. For example, in Figure 6.1b, different cell colonies ($\mathcal{Q}$) are placed on the chip. (2) In the second step, the soluble compounds are inserted using either the top-to-bottom route (insertion from the top inlets: $In1$ to $In8$ in Figure 6.1b, exit using the bottom outlets) or the left-to-right route. When a compound is inserted from an inlet (e.g., $In2$), it traverses through all colonies in its way till it reaches the outlet (e.g., the colony set (Q^-, Q_9, Q_7, Q_1, Q_6, Q_1, Q_6, Q^+) in the second column in Figure 6.1b), influencing all 8 experiments (every cell colony is a separate experiment). The soluble compounds are manually placed in inlet reservoirs (see reservoirs in Figure 6.1a). However, they are transported to the biochip automatically using pumps according to the insertion schedule determined offline, i.e., the schedule is decided before the experiment is performed. The pump can fetch from any desired inlet reservoir. The pumps are controlled using a programmable custom-made electrical controller. The cell placement pattern (1) and the compound insertion schedule (2) determine the number of unique experiments in an experimental stage. The serialized insertion of compounds (i.e., a compound inserted from an inlet affects all colonies placed between that inlet and the designated outlet) reduces the probability of having a high number of unique experiments.

Today the experimental design, i.e., deciding the placement pattern of the cell colonies on the biochip and the schedule of the stimuli insertion (which compounds to insert, in what sequence, and from which inlets), is done manually resulting in undesired repetitions of experiments on the chip, reducing the overall throughput. We propose an optimization approach to automate the cell culture biochip experimental design such that the experimental throughput is maximized. Maximizing the throughput increases system productivity, saving time (one cell culture experiment can take days to complete) and reducing costs; the purified proteins and compounds used in the experiments are highly expensive.

6.1 System Model

6.1.1 Biochip Architecture

Figure 6.1b gives a high-level representation of a biochip architecture [76] for cell culturing. The biochip itself is one big chamber (imagine a very wide microfluidic channel) with 8 inlets at the left and 8 at the top, and 8 outlets at the right and 8 at the bottom. The biochip uses the laminar flow property [86] which is defined as the flow of fluids in parallel layers without any disruption between the layers. This means that when different compounds are pushed in from different inlets (e.g., from $In1$ to $In8$), because of the microscale volumes they flow in

parallel without mixing thus creating 8 parallel flows. Similar effect is obtained when inlets on the left ($In9$ to $In16$) are used. Only one set of inlets can be used at a time. The biochip is represented as an $A_R \times A_C$ matrix (A_R rows and A_C columns, e.g., for the chip in Figure 6.1b, $A_R = A_C = 8$). Each element of the $A_R \times A_C$ matrix hosts a cell colony. The off-chip valves V1 and V2 control the flow to the outlets.

Before starting an experiment, the locations on the chamber where the cell colonies are to be placed are tagged using DNA spotting technique. Then, to carry out the experiment, first, valve V2 is opened and valve V1 is closed, allowing different cell types to be pumped into the large chamber. Due to laminar flows, eight corresponding stripes of cells are produced. After sedimentation, cells adhere to the bottom of the chip according to the DNA spotting. Then, valve V2 is closed and valve V1 is opened, allowing eight soluble compounds to perfuse over the cells in the perpendicular direction creating simultaneous experiments in the chip. It is also possible to insert soluble compounds using the same inlets from which the cells were initially inserted (valve V1 closed, V2 open), but only one type of inlets can be open at a time.

Two special types of cell colonies, positive (Q^+) and negative (Q^-) controls (selected based on the application being executed on the chip, i.e., depending on the cell colonies and the input compounds), are used to ensure the quality of compounds being inserted into the chamber and to guarantee that the conditions in the chamber are uniform [77]. The actual size of the chip here is 8×8, but since the controls need to be placed at the boundary locations, therefore the active area (on which the cell colonies under test are placed) is reduced to 6×6.

As mentioned, the biochip shown in Figure 6.1b has one large chamber and relies on the laminar flow properties of the liquids for ensuring that the inserted cells and compounds do not mix with each other. This is prone to different faults, e.g., if an air bubble enters the chip from one of the inlets, it will not just affect the cell colonies in front of the inlet it entered from but instead all cell colonies on the chip can become enclosed in the air bubble. This will result in the failure of of all experiments on the chip.

Another option would be to isolate the rows and columns in the chip, i.e., have independent chambers instead of one large chamber. The mVLSI technology provides such an option offering more control, as is the case with the biochip shown in Figure 6.2. The biochip shown in the figure is shaped in the form of a 20×20 matrix and is used to perform 400 distinct PCR reactions (i.e., it is not used for cell culturing) [51]. Each vertex in the matrix contains a reactor, so there are 400 reactors (20×20) in total. Each row of reactors is connected to a separate input port through which primers may be loaded. Each column can similarly load the reactors with different DNA templates [51].

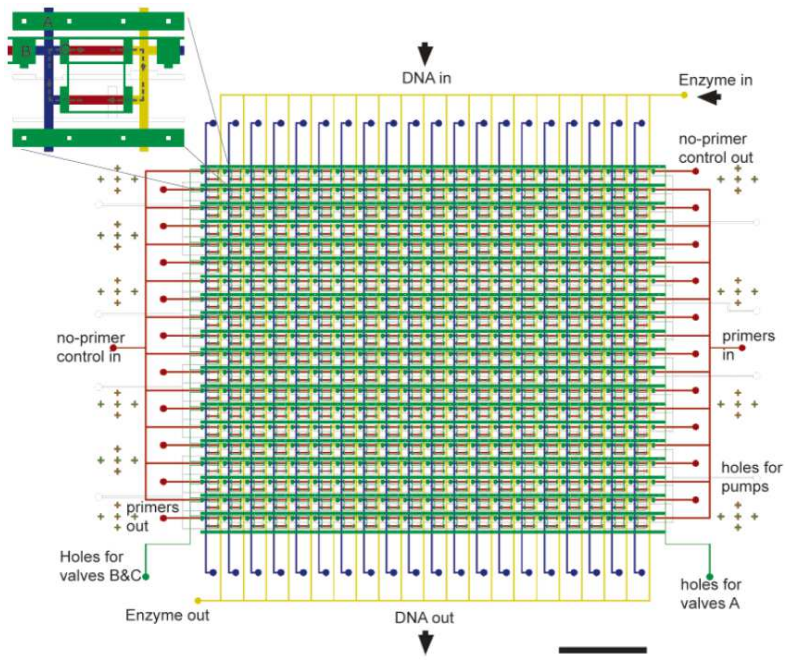


Figure 6.2: PCR Biochip with $A_R = A_C = 20$ (Scale Bar 6.4 mm) [51]

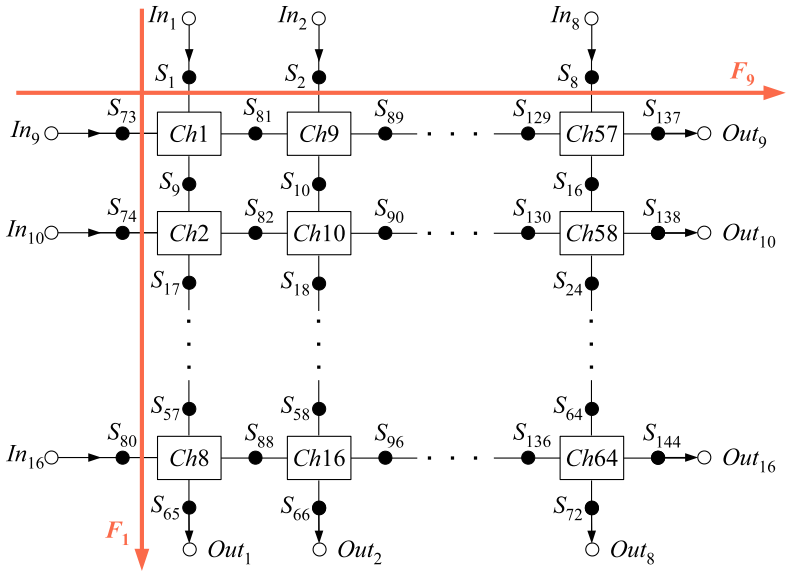


Figure 6.3: Cell Culture Chip: Schematic View

Table 6.1: Flow Path Set ($\mathcal{F}$) and the Routing Constraints ($\mathcal{K}$)

Flow Path Set	Routing Constraints
$F_1 = (In_1, S_1, Ch_1, S_9, Ch_2, S_{17}, Ch_3, S_{25}, Ch_4,)$	K_1: (K_9 to K_{16})
$S_{33}, Ch_5, S_{41}, Ch_6, S_{49}, Ch_7, S_{57}, Ch_8, S_{65}, Out_1)$	...
$F_2 = (In_2, ..., Out_2)$	K_8: (K_9 to K_{16})
...	K_9: (K_1 to K_8)
$F_{15} = (In_{15}, ..., Out_{15})$	...
$F_{16} = (In_{16}, ..., Out_{16})$	K_{16}: (K_1 to K_8)

The chip has 2,680 valves in total, placed horizontally and vertically. Since the flows are activated either vertically (top to bottom) or horizontally (right to left) therefore all these valves are controlled by only two control pins connected to two independent pressure sources, i.e., all valves required to be closed for vertical flows are controlled by one pressure source since they are always closed at the same time. Same applies to the valve activation for horizontal flows [51].

Figure 6.3 shows the schematic view of the mVLSI architecture that we consider for a cell culture biochip, based on the architecture discussed above (Figure 6.2). The architecture shown has 64 chambers (8×8) for placing 64 cell colonies (same number of cell colonies as for the chip in Figure 6.1b). This architecture offers isolated chambers and more control. The chip uses a network of 144 switches to control 64 cell culture chambers. The flow paths and the associated routing constraints are shown in Table 6.1. Switch set $\{S_{73}$ to $S_{144}\}$ is closed for executing flow paths F_1 to F_8 and switch set $\{S_1$ to $S_{72}\}$ is closed for executing flow paths F_9 to F_{16}. Therefore all 144 switches can be controlled using only 2 control pins, i.e., only 2 external pressure sources are needed. We consider that the chip shown in Figure 6.3 is used in the platform in Figure 6.1b.

6.1.2 Experimental Design

As stated earlier, exposure of a cell colony to a sequence of soluble compounds and monitoring its reaction is termed as an experiment. All the experiments being carried out in a chip are collectively considered as one experimental stage. In order to explain the experimental design process, we consider a 6×6 cell culture biochip that has an active area of 4×4. The chip is used to carry out experiments in $n = 3$ experimental stages, Figure 6.4 (a) to (c).We consider a set $\mathcal{Q}$ of two cell colonies (Q_1 and Q_2) and a set $\mathcal{J}$ of three compounds to be inserted (J_1 to J_3). We would like to expose the cells to the compounds, in any order. The exposure time per compound is fixed. However, since we allow the compound to be repeated in an exposure sequence, we can thus increase the

exposure time for the same compound. The maximum number of experimental stages to be used is generally decided based on the available budget (experiments are highly expensive).

The placement of cell colonies on the biochip is given by the matrix $\mathcal{Z}$; see for example Z_1 in Figure 6.4a. The sequence of compounds is captured by the schedule $\mathcal{Y}$, composed of $< Y_R, Y_C, Y_Z >$. The compounds can be inserted either from left to right (row-wise) or from top to bottom (column-wise). Two matrices Y_R and Y_C represent the compound placement at the insertion point, row-wise and column-wise, respectively. The set Y_Z represents the sequence in which the compounds are inserted. A "1" in Y_Z marks a column-wise insertion whereas a "0" represents a row-wise insertion, see for example $Y_1 = < Y_R^1, Y_C^1, Y_Z^1 >$ in Figure 6.4a.

Through experiments, we have to find out the sequence of compounds that provide the desired result for a specific cell colony. To maximize the number of combinations applied to a cell colony, it is important to perform as many unique experiments on the chip as possible. A unique experiment is one which is not repeated (in another part of the chip, or during another experimental stage).

Consider the cell colony Q_2 placed in the top left corner in Figure 6.4a. The number of compounds to which the colonies are to be exposed is considered to be fixed to 3 by the biologist. Y_Z^1 in Figure 6.4a is {1, 0, 1}. This means that the first compound insertion will be made from Y_C, the second from Y_R and the third again from Y_C. Therefore, Q_2 placed at the top left corner will be exposed to the following compounds: J_1 (from Y_C), J_3 (from Y_R), J_2 (from Y_C), denoted as "132". There are 16 colonies placement location on the chip, 8 are filled with Q_1 and 8 with Q_2. Based on the schedule $Y_1 = < Y_R^1, Y_C^1, Y_Z^1 >$ given in Figure 6.4a, following experiments are performed (sequence of the experiments listed below is a row by row listing from the matrix):

$$\begin{aligned} Q_1 &: 233, 331, 122, \underline{122}, 112, \underline{112}, \underline{233}, \underline{331} \\ Q_2 &: 132, \underline{132}, 223, 321, 213, 311, \underline{132}, \underline{132} \end{aligned} \tag{6.1}$$

where we have underlined the experiments that have been repeated. Repetition of experiments results in inefficient utilization of the chip. This results in diminished system productivity and increased expenses, both in terms of time and cost. Therefore, efficient experimental design is essential. Note that although our model uses two matrices, Y_R and Y_C to capture the compounds, only one such matrix is used in the implementation. The set Y_Z will determine if a particular row in this matrix is fed row-wise or column-wise.

Considering an $A_R \times A_C$ biochip, the maximum number of unique experiments that can be carried out in one experimental stage is given by the active chip

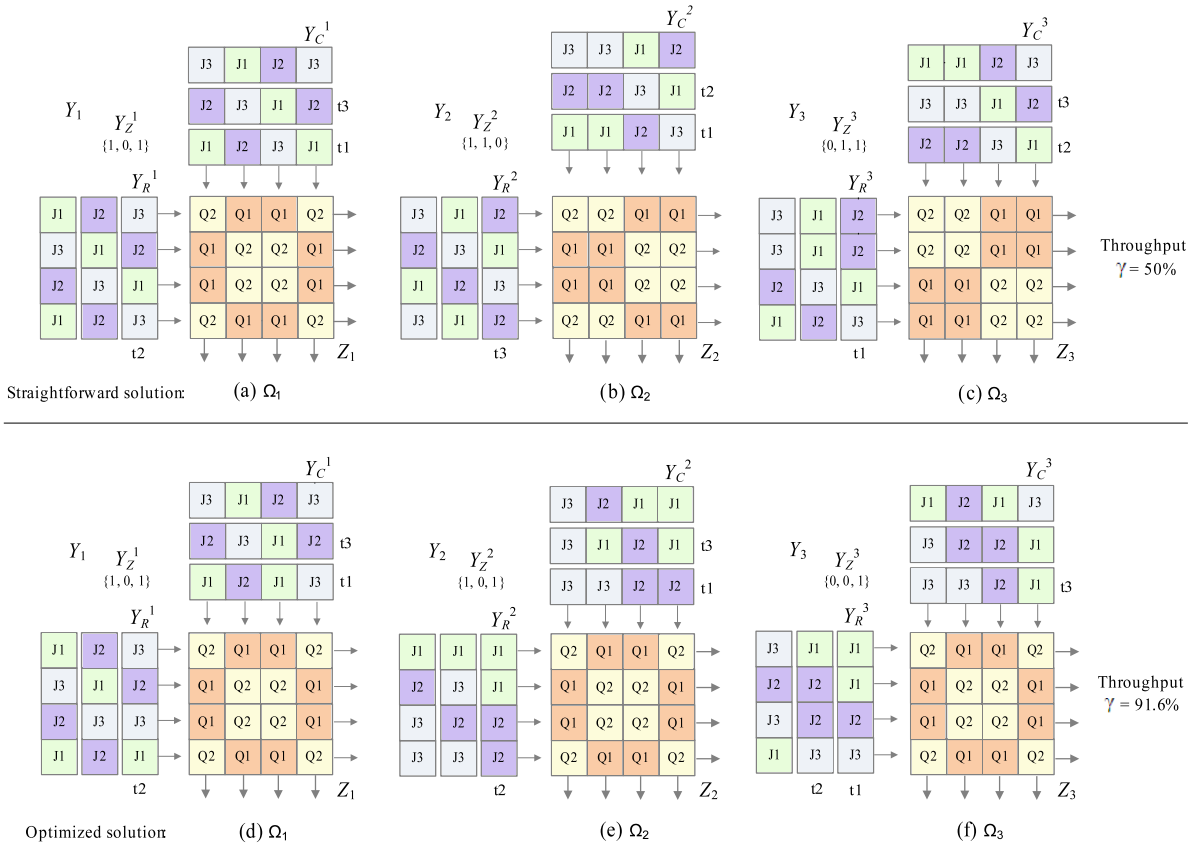


Figure 6.4: Motivational Example

area: $A'_R \times A'_C$, where $A'_R = (A_R - 2)$ and $A'_C = (A_C - 2)$. Therefore, for Figure 6.4a, the maximum utilization U_{max} of the biochip is: $U_{max} = 4{\times}4 = 16$. The maximum number of combinations possible for a given colony depends on the experimental specifications, captured by the set of compounds $\mathcal{J}$ and the number of compounds I in an exposure sequence and is equal to $(|\mathcal{J}|^I)$, where $|\mathcal{J}|$ is the number of elements in the set $\mathcal{J}$. For the experiments in Figure 6.4a, $(|\mathcal{J}|^I) = 27$, the total number of combinations of $I = 3$ items taken from a set of three elements $\mathcal{J} = J_1, J_2, J_3$. The number of possible compound combinations $(|\mathcal{J}|^I)$ for a certain cell colony is typically larger than the maximum chip capacity U_{max}, which has to be divided among several cell colonies. For the placement in Figure 6.4a, colony Q_1 can use only half of the total capacity, U_{max} / 2 = 8. Therefore, to achieve a wider coverage from the set of 27 unique experiments per colony, it is imperative to have more experimental stages even if the chip is fully utilized for the first stage, i.e., all 16 experiments (8 per colony) carried out in the first stage are unique. In the current case, the biologist has specified 3 experimental stages. This means that in the ideal case, 24 out of the 27 unique experiments can be achieved for each colony. The same biochip may be washed and manually reloaded with cell colonies for reuse in multiple experimental stages.

6.2 Problem Formulation

The problem we are addressing in this chapter can be formulated as follows. Given (1) a cell culture biochip architecture model consisting of an $A_R \times A_C$ matrix representing the cell culture chip chambers, (2) a set of compounds $\mathcal{J}$ that are to be inserted from the inlets, (3) a set of cell colonies $\mathcal{Q}$ that are to be placed on the cell culture chambers, (4) the number of compounds per exposure sequence I, and (5) the total number of experimental stages n to be performed, we are interested in designing each experimental stage Ω_i, $i = 1...n$, such that the experimental throughput γ is maximized.

Designing an experimental stage $\Omega_i < Z_i, Y_i >$ means deciding for each stage i on: (1) the placement Z_i of the cell colonies from the set $\mathcal{Q}$ and (2) the compound exposure schedule $Y_i = < Y_R^i, Y_C^i, Y_Z^i >$. The throughput of a cell culture biochip is defined as:

$$\gamma = \frac{|\mathcal{E}_U|}{min(n \times U_{max}, (|\mathcal{J}|)^I \times |\mathcal{Q}|)} \tag{6.2}$$

where $|\mathcal{E}_U|$ is the number of elements in the set $\mathcal{E}_U$ of unique experiments over n stages, considering the synthesized design Ω. The maximum number of unique experiments is bounded by (i) the chip capacity, $n \times U_{max}$, and by (ii) the total

number of possible combinations, considering all cell colonies $\mathcal{Q}$, $(|\mathcal{J}|^I) \times |\mathcal{Q}|$, whichever of the two values is smaller. Therefore, γ represents a percentage of unique experiments obtained through our design Ω to the maximum possible, given the chip capacity or the number of combinations. Note that a 100% throughput might not be possible to obtain, and we have no way of knowing maximum possible throughput unless we obtain the optimal design.

Let us use the example in Figure 6.4 to illustrate our problem. In Figure 6.4, we have $A'_R = A'_C = 6 - 2 = 4$, $\mathcal{Q} = Q_1, Q_2$, $\mathcal{J} = J_1, J_2, J_3$, $I = 3$ and $n = 3$. Let us suppose we start from the configuration shown in Figure 6.4a, where we assume that the placement Z_1 and schedule Y_1 have been decided as depicted in the figure. All the experiments obtained in this first stage $\Omega_1 = < Z_1, Y_1 >$ have been listed in eq. 6.1. As shown, for Q_2 5 out of the total 8 experiments carried out are unique and for Q_1 only 4 experiments are unique. The remaining experiments (underlined) are repetitions. Therefore, the chip capacity for this stage is poorly utilized.

In order to increase to number of unique experiments we have to carefully design the second stage $\Omega_2 = < Z_2, Y_2 >$. For the second stage, the same biochip can be used but, as stated earlier, it needs to be washed and reloaded with cell colonies at the decided placements. A simple way to generate a stage Ω_i is to modify a previous stage Ω_{i-1} using a certain set of rules. We have decided to roll over the placement Z_{i-1} and the contents of the schedule Y_{i-1} in the hope to increase the number of unique combinations. Thus, we obtain the stage design Ω_2 given in Figure 6.4b, where Z_1, Y_R^1, Y_C^1 and Y_Z^1 from Figure 6.4a are all rolled to the right, i.e., all the contents are shifted right and the ones in the right most position are moved to the first entries on the left. The experiment is run again and the following experiments are generated:

$$\begin{aligned} Q_1 &: 232, 312, 121, \underline{121}, 123, \underline{123}, \underline{232}, \underline{312} \\ Q_2 &: 122, \underline{122}, 231, \mathbf{311}, 233, 313, \underline{122}, \underline{122} \end{aligned} \tag{6.3}$$

The experiments underlined are repeated in the same stage and the ones marked in bold have already been covered in the previously conducted experimental stage listed in eq. 6.1. Each cell colony, Q_1 and Q_2, now has 4 more unique experiments, whereas ideally, 8 unique experiments could have been achieved in this experimental stage. Again, the chip capacity has been utilized poorly for this stage as well and needs to be improved.

More variations are now required in the experimental settings in order to increase the probability of covering more unique experiments. Thus, for the third experimental stage, a top-to-bottom roll is performed on the settings in Figure 6.4b, i.e., all the contents are shifted one step towards the bottom and the ones at the bottom position are moved to the first entries on the top. Since the

top-to-bottom roll cannot be performed on the Y_Z^2 (1-D array), it is rolled to the right instead. The new settings are shown in Figure 6.4c. The generated experiments for these new settings are:

$$\begin{aligned} Q_1 &: 231, 212, \underline{231}, \underline{212}, \mathbf{123}, \mathbf{123}, 323, \underline{323} \\ Q_2 &: \mathbf{223}, \mathbf{223}, \mathbf{223}, \mathbf{223}, 131, 112, 331, 312 \end{aligned} \tag{6.4}$$

As shown above, only 3 new unique experiments for Q_1 and 4 for Q_2 have been generated and the remaining experiments are repetitions.

For all three experimental stages listed above, the joint experimental throughput γ can be calculated using eq. 6.2. For the cell colony Q_1, the number of unique experiments in Stage1 (eq. 6.1), Stage2 (eq. 6.3) and Stage3 (eq. 6.4) are 4, 4 and 3 respectively. For Q_2, these are 5, 4 and 4. Sum of all these unique experiments,$(4+5+4+4+3+4) = 24$, is the numerator of eq. 6.2 when calculating the throughput. For the denominator, $n = 3$, $U_{max} = 16$, $|\mathcal{J}| = 3$, $I = 3$, and $|\mathcal{Q}| = 2$. The throughput, therefore, is equal to $(4+5+4+4+3+4)/\ min(3\times16, 3^3\times2) = 24/\ 48 = 50\%$. Therefore, only half of the system experimental capacity is being utilized.

A lower chip throughput translates into reduced system productivity and wastage of resources. We are interested in designing an optimized experimental design Ω such that the throughput γ is maximized. Such an optimized Ω is presented in Figure 6.4(d–f), where the throughput obtained is 91.6% instead of 50%. Figure 6.4(d–f) show the settings for the three experimental stages designed for higher throughput. The experiments generated using these settings are:

$$\begin{aligned} Stage1 \quad & Q_1 : 233, 131, 122, 322, 132, 332, 213, 111 \\ & Q_2 : 132, 332, 223, 121, 233, 131, 112, 312 \\ Stage2 \quad & Q_1 : 311, 212, 313, 211, 323, 221, 321, 222 \\ & Q_2 : 313, 211, 311, 212, 321, 222, 323, 221 \\ Stage3 \quad & Q_1 : 113, 112, 123, 121, 223, \mathbf{221}, 333, \mathbf{332} \\ & Q_2 : 113, 111, 123, 122, \mathbf{223}, \mathbf{222}, 333, 331 \end{aligned} \tag{6.5}$$

resulting in $\gamma = (8+8+8+8+6+6)\ /\ 48 = 91.6\%$.

Finding such a solution, even for this simple case, can be quite tedious and complex. The next section presents our optimization strategy for automatically deriving the experimental design Ω which maximizes the throughput γ.

6.3 Experimental Throughput Optimization

The problem presented in the previous section is NP-complete. To maximize the throughput γ over all the n stages, we would have to simultaneously optimize all the matrices $\Omega_i = < Z_i, Y_i >$ for all stages $i = 1...n$, since the uniqueness of an experiment is defined across all stages. Instead, our Experimental Throughput Optimization (ETO) strategy, presented in Figure 6.5, is to optimize the stages incrementally, one stage at a time. Therefore, when optimizing a stage Ω_i, we aim to maximize the number of unique experiments considering all the experiments generated in stages Ω_1 to Ω_i. This approach does not guarantee to find the optimal experimental design, but as the evaluation in Section 6.4 shows, it can obtain very good results in a short time.

The initial solution is generated using a heuristic presented in Section 6.3.1 (line 1 in Figure 6.5). The final stages are generated iteratively (lines 3–6 in Figure 6.5) using an approach based on the Simulated Annealing (SA) metaheuristic (presented in detail in Section 6.3.2). SA takes as input the configuration of the previous stage $< Z_{i-1}, Y_{i-1} >$ and the set of unique experiments $\mathcal{E}_U$ generated so far, and determines the configuration $\Omega_i = < Z_i, Y_i >$ such that the number of unique experiments generated are maximized. The set $\mathcal{E}_U$ contains all the unique experiments generated, from Ω_1 to Ω_{i-1}.

6.3.1 Initial Solution

The initial configuration is created using a Descend-Ascend scheme. Figure 6.6 shows the initial solution for a 4×4 chip hosting 6 colonies, to be inserted with a total of 4 compounds and having 3 compounds per exposure sequence. The configuration starts by placing the colony/ compound with the highest value identifier at the top left location. Then, the second highest identifier (descend)

ETO(A_R, A_C, $\mathcal{Q}$, $\mathcal{J}$, I, n)
1 $< Y_0 , Z_0 >$ = InitialSolution(A_R, A_C, $\mathcal{Q}$, $\mathcal{J}$, I)
2 $E_U = \emptyset$
3 **for** $i = 1 \rightarrow n$ **do**
4 $< Z_i, Y_i, E_{Ui} >$ = SimulatedAnnealing (Z_{i-1}, Y_{i-1}, E_U)
5 $E_U = E_U \cup E_{Ui}$
6 **end for**
7 **return** $\Omega = < \mathcal{Z}, \mathcal{Y} >$

Figure 6.5: Optimization Strategy

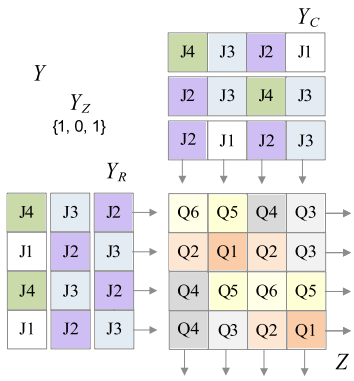


Figure 6.6: Initial Solution

is placed at the next row position and so on. At the end of each row, the sequence moves to the start of the next row. On reaching the smallest value identifier, the scheme switches to ascend, i.e., the identifier placement is now made in the ascending order.

6.3.2 Simulated Annealing

Simulated Annealing (SA) (introduced in Section 4.5.2.1) is an optimization metaheuristic inspired from the annealing process in metallurgy and is a variant of the neighborhood search technique. This is our first attempt to solve the problem of optimizing the experimental throughput, therefore we have decided to use SA because of its simplicity. The right choice of optimization approach depends on the particular problem and we plan to investigate which is the best approach in our future work.

SA uses design transformations ("moves") to transform the current solution (Z_{now}, Y_{now}) in order to explore the design space. There are three types of moves: (i) moves that change the inputs Y_R and Y_C, (ii) moves that change Y_Z, i.e., whether to give input from Y_R or Y_C, and (iii) moves that change placement $\mathcal{Z}$ of the colonies on the chamber. The moves are explained in more detail below:

- For changing the matrix Y_C of $I \times A'_R$ elements, we randomly select two elements (compounds J_i and J_j) and swap them. These are the type of moves performed most often, since they have a large impact on the generation of new combinations of compounds applied to the cell colonies. Similar moves are performed on Y_R.

- Moves that transform Y_Z are applied when the value of the temperature goes below 1, significantly reducing the acceptance probability of the deteriorated cost solutions. The temperature is then reset and a random move in Y_Z is made. Since Y_Z has only 0 or 1 as member elements, the maximum number of unique moves is limited.
- Moves in $\mathcal{Z}$ are applied the least often, only when moves in Y_Z have been exhausted. A move in the $A'_R \times A'_C$ matrix $\mathcal{Z}$ is performed by randomly selecting the elements ($Q_i \neq Q_j$) and swapping them with each other.

The algorithm stops if either the maximum number of unique experiments obtained in one stage considering the biochip capacity is found or the termination time is reached.

6.4 Experimental Evaluation

In order to evaluate our Experimental Throughput Optimization (ETO) approach, we have performed two sets of experiments. The algorithms were implemented in C++, running on Lenovo T400s ThinkPad with Core 2 Duo Processors at 2.53 GHz and 4 GB of RAM.

For comparison purposes during evaluation, we have also implemented a straightforward (SF) approach. This is an approach that a good engineer may use when no optimization tools are available. SF starts from an initial solution obtained with the Descend-Ascend scheme given in Section 6.3.1 and then performs a left-to-right roll on the placement $\mathcal{Z}$ and the contents of the schedule $\mathcal{Y}$ in the hope to increase the number of unique combinations, e.g., in Figure 6.4b we obtain the stage design Ω_2 by rolling over the settings in Figure 6.4a. Next, a top-to-bottom roll is performed. Since the top-to-bottom roll cannot be performed on Y_Z, it is rolled to the right instead, as shown in the transition from the settings in Figure 6.4b to Figure 6.4c. For each experimental stage, SF alternately performs a left-to-right and top-to-bottom roll, until termination criteria is reached. On termination, SF returns the best solution found, i.e., the one that maximized the system throughput.

In the first set of experiments shown in Table 6.2, we are interested in evaluating the quality of ETO in terms of its ability to maximize the experimental throughput across all desired stages. We have used a total of 9 experimental settings as presented in column 1 to 4 in Table 6.2 ranging the experimental chip area from 6×6 to 14×14, the number of compounds from 2 to 8, the number of cell colonies from 2 to 9 and the number of experimental stages from 2

Table 6.2: Full Factorial Design

$A'_R \times A'_C$	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{Q}\rvert$	n	SF γ (%)	Best ETO γ (%)	Average ETO γ (%)	Standard Deviation
6×6	2	9	2	50	83.3	78.7	3.46
	3	7	6	43.9	88.4	84.2	3.04
	4	5	9	27.8	82.8	77.8	3.43
10×10	4	3	2	12.5	78.1	74.3	2.89
	5	2	3	42.4	85.2	78.4	2.57
	6	3	7	48.3	74.8	71.8	2.17
14×14	6	3	4	44.9	76.2	73.5	1.72
	7	4	7	22.4	68.9	66.8	1.36
	8	4	11	46.5	69.8	68.3	1.12

to 11. The number of compounds per exposure sequence I was set to 3 for all cases. We have determined the combinations of parameters (each row in the table) such that all possible combinations of compounds for each cell colony may be achievable within the imposed chip area and the number of stages. It is important to note here that even if the chip capacity is equal to (or more than) the maximum possible combinations of compounds for all cell colonies placed on the chip, 100% throughput might still not be achievable. This is because the compounds that we feed into the chip affect an entire row (or column), therefore there might be situations where a schedule and placement that would guarantee 100% throughput across all stages may not exist.

Table 6.2 gives the throughput γ achieved by SF and ETO. ETO generated solutions provide experimental throughput as high as 88% and it does not go below 65% in any of the cases. As shown, ETO performs significantly better than SF. Together with the best solution, Table 6.2 also presents the average and the standard deviation obtained after 10 runs of ETO, exploring the solution space differently in every run. As presented, the standard deviation is quite small indicating that ETO consistently finds solutions that are close to the best solution. We have used a time limit of 10 minutes for ETO.

We have also evaluated our proposed approach for the case when the maximum possible number of combinations exceeds the capacity of the available number of chips. A real world example would be that of the fractionally factorial experimental design performed in the first phase of drug discovery. Table 6.3 presents the results. We have considered an active biochip area of 6×6 with $n = 3$ stages and have progressively increased the number of combinations by varying the number of compounds $|\mathcal{J}|$ and the number of cell colonies $|\mathcal{Q}|$. I was set to 5

Table 6.3: Fractionally Factorial Design

		SF		ETO	
$\|\mathcal{J}\|$	$\|\mathcal{Q}\|$	$\|E_U\|$	γ (%)	$\|E_U\|$	γ (%)
$A'_R \times A'_C = 6{\times}6$, $n = 3$, I = 5					
2	5	30	27.7	89	82.4
2	6	36	33.3	97	89.8
2	7	36	33.3	102	94.4
2	8	36	33.3	104	96.2
2	9	54	50	106	98.1
2	10	60	55.5	106	98.1
3	2	36	33.3	108	100
3	3	36	33.3	108	100

for all cases. As we can see from Table 6.3, ETO can get close to the 100% throughput in most of the cases, which means that the biochip is utilized to its full capacity over all the n stages.

In order to determine the quality of our SA-based ETO strategy, we have used an exhaustive search to determine the optimal solutions. Since the runtime of the exhaustive search is prohibitively large, we were only able to run it for smaller examples, lines 1, 4 and 7 in Table 6.2. In these cases, our ETO approach is capable of obtaining solutions which are very close to the optimum. For the experimental setups in lines 1, 4 and 7, the difference in terms of the cost function is only 1.4%, 2.1% and 2.3%, respectively.

6.5 Summary

In this chapter, we have presented a Simulated Annealing based approach in order to design high throughput experiments for cell culture microfluidic biochips. The proposed approach considers multiple parameters (e.g., chip size, number of cell colonies, and number of soluble compounds) as inputs and generates design settings (placements and schedules) for the desired number of experimental stages such that the system throughput is maximized. Multiple experimental setups have been used for evaluating the effectiveness of the proposed approach. We have shown that by optimizing the experimental design, significant improvements in the experimental throughput (and chip utilization) can be achieved, increasing the system productivity, saving time and reducing costs.

Chapter 7

Conclusions and Future Work

This chapter presents the conclusions and possible future extensions of the work presented in this thesis.

7.1 Conclusions

In this thesis we have proposed, for the first time to our knowledge, a top-down design methodology for the flow-based mVLSI biochips. The current design practice for these chips is primarily manual. With the fabrication technology for the mVLSI biochips, soft lithography, advancing faster than Moore's law (densities approaching 1 million valves per cm^2), top-down methodologies and design automation tools are absolutely essential in order to manage the design complexity and exploit full potential of these devices.

We have proposed models for the biochip architecture as well as the microfluidic components. Using these models, we have proposed the framework for the application-specific architectural synthesis of these chips, mapping of biochemical applications onto the mVLSI architectures and generation of optimized schedules as well as control synthesis for these chips in order to automatically execute the applications on mVLSI architectures. We have also proposed pin count minimization schemes to enhance chip scalability and throughput maximization

techniques to increase chip productivity. We have extensively evaluated our approach using real-life assays as well as synthetic benchmarks.

The primary conclusion of this thesis is that by using the proposed models, top-down design automation methods for the mVLSI biochips can be successfully formulated. This has been shown using the proposed synthesis frameworks. The proposed methods are expected to facilitate programmability and automation, resulting in full utilization of the mVLSI potential and emergence of a large biochip market. The conclusions of the individual chapters are summarized below:

- In Chapter 2, we have proposed a dual-layer modeling framework for the mVLSI components. We have shown how the model captures the component operational phases at the flow layer as well as the control layer valve activations needed to execute these operational phases. We have proposed a topology graph-based model for the mVLSI biochips and shown how it can capture the chip components, their interconnections, the fluid flow paths on the chip and also the routing constraints.

- In Chapter 3, we have formulated the problem of mapping the biochemical application onto the mVLSI architectures using the models proposed in Chapter 2. First, we have shown how the problem of mapping the biochemical operations onto the mVLSI biochips can be solved optimally (in terms of application completion time) using a constraint programming (CP) framework. Since the CP approach turns out to be computationally very intensive, we have then proposed a binding and scheduling heuristic (based on List Scheduling) that performs the operation binding and scheduling as well as fluidic routing. We have shown that the heuristic approach produces good results in short time. This is the first time that an application mapping approach has been proposed for the mVLSI biochips.

- In Chapter 4, a new problem of application-specific architectural synthesis has been formulated. Our proposed solution strategy starts by allocating the required components from the library, then generates the schematic and finally, performs the physical synthesis (placement and routing) of the chip. We have shown how the algorithms from the VLSI domain can be tailored and applied to the mVLSI biochips. This is the first time that an application-specific architectural synthesis framework has been proposed for the mVLSI architectures.

- We have proposed a top-down control synthesis approach for implementing the biochemical applications on the mVLSI architecture in Chapter 5. We have shown how, using the proposed models, control logic (valve activation sequence for executing the application) can be automatically synthesized.

The high pin count of mVLSI chips is a bottleneck to the chip scalability. We have shown how optimization schemes can be used to minimize the pin count, enhancing scalability and reducing the macro-assembly around the chip.

- Chapter 6 deals with the cell culture biochips. These chips perform experiments that take weeks to complete and are highly expensive. We have proposed models for the cell culture chips and have shown how the experimental design can be optimized, such that the number of experiments can be maximized enhancing the chip productivity.

7.2 Future Work

The thesis has presented top-down design methods for the mVLSI biochips. The work can be extended in many ways in order to address the remaining challenges in the mVLSI domain. Some possible extensions are as follows:

- As discussed in Section 1.4, the primary design objective of this thesis has been the application completion time minimization. Further research can be carried out using other objectives, e.g., pin-constrained mVLSi design.
- The control pin minimization (presented in Chapter 5) is done by sharing the control pins between multiple valves. The minimization scheme reduces the pin count but does not guarantee a routable control layer. This is because only two layers are available for performing the control layer routing and the control channels are not allowed to intersect, limiting the routability. A feedback system can be used between the pin count minimization and the control layer physical synthesis, in order to guarantee a routable control layer.
- Since the synthesis strategies are designed based on the requirements captured by the application model, more research can be carried out on the application model itself in order to ensure that it accurately captures the application requirements. For example, an application may impose a timing constraint that two fluids must be mixed within x number of seconds after being heated. This kind of constraint is not captured by the current application model.

Appendix A

List of Notations

Table A.1: List of Notations

Notation	Description
ψ	Mapping implementation
η	Control logic
$\delta_{\mathcal{G}}$	Application completion time
γ	Biochip throughput
$\mathcal{E}_U$	Unique experiments
U_{max}	Maximum chip capacity
$\mathcal{G}_C$	Control coloring graph
$\mathcal{V}_C$	Vertices in $\mathcal{G}_C$
$\mathcal{E}_C$	Edges in $\mathcal{G}_C$
$\mathcal{A}$	Biochip architecture model
$\mathcal{B}$	Binding function
C	Execution time
$\mathcal{D}$	Set of directed edges
$\mathcal{E}$	Set of edges in $\mathcal{G}$
$\mathcal{F}$	Set of flow paths
$\mathcal{G}$	Biochemical application model
H	Geometrical dimensions of a component
I	Number of compounds/ exposure sequence

$\mathcal{J}$	Set of compounds
$\mathcal{K}$	Set of routing constraints
$\mathcal{L}$	Component library
$\mathcal{M}$	Set of components
$\mathcal{N}$	Set of vertices in $\mathcal{A}$
$\mathcal{O}$	Set of operations in $\mathcal{G}$
$\mathcal{P}$	Operational phases in a component
$\mathcal{Q}$	Set of colonies
$\mathcal{R}_f$	Flow layer routing
$\mathcal{R}_c$	Control layer routing
$\mathcal{S}$	Set of switches
T	Temperature
$\mathcal{U}$	Component allocation
w_c	Capacity weight
w_v	Volume weight
$\mathcal{X}$	Schedule
$\mathcal{Y}$	Component insertion schedule
$\mathcal{Z}_f$	Flow layer placement
$\mathcal{Z}_c$	Control layer placement
$\mathcal{Z}$	Cell colony placement

Bibliography

[1] Affymetrix Inc. http://www.affymetrix.com/.

[2] Agilent Technologies. http://www.home.agilent.com/.

[3] AutoCAD Products. http://usa.autodesk.com/autocad-products/.

[4] Biochips: A Global Strategic Business Report. http://www.strategyr.com/Biochips_Market_Report.asp.

[5] Caliper Life Sciences Inc. http://www.caliperls.com/.

[6] Fluidigm: Chips and Kits. http://www.fluidigm.com/chips-kits.html.

[7] Fluidigm Corporation. http://www.fluidigm.com/.

[8] GE Healthcare Ltd. http://www.gehealthcare.com/.

[9] Illumina, Inc. http://www.illumina.com/.

[10] International Technology Roadmap for Semiconductors 2011 Edition. http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf.

[11] ISI Web of Science, search for topic "microfluidic". http://www.isiknowledge.com.

[12] KNI Microfluidic Foundry - CalTech. http://www.kni.caltech.edu/foundry/.

[13] Life Technologies Corporation. http://www.lifetechnologies.com/.

[14] mLSI Biochips. https://sites.google.com/site/mlsibiochips/.

[15] Stanford Microfluidic Foundry. http://www.stanford.edu/group/foundry/.

[16] Stephen Quake's Group at Stanford University. `http://thebigone.stanford.edu/`.

[17] United States Patent and Trademark office, search issued patents for "microfluidic" in title or abstract. `http://patft.uspto.gov`.

[18] Verinata Health. `http://www.verinata.com/`.

[19] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson. Aquacore: A programmable architecture for microfluidics. In *Proceedings of the 34th annual international symposium on Computer architecture*, pages 254–265, 2007.

[20] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson. Automatic volume management for programmable microfluidics. In *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation*, 2008.

[21] N. Amin, W. Thies, and S. Amarasinghe. Computer-aided design for microfluidic chips based on multilayer soft lithography. In *Proceedings of the IEEE International Conference on Computer Design*, 2009.

[22] V. Ananthanarayanan and W. Thies. Biocoder: A programming language for standardizing and automating biology protocols. *Journal of Biological Engineering*, 4(13), 2010.

[23] I. E. Araci and S. R. Quake. Microfluidic very large scale integration (mvlsi) with integrated micromechanical valves. *Lab Chip*, 12:2830–2806, 2012.

[24] K. Chakrabarty and T. Xu. *Digital Microfluidic Biochips: Design Automation and Optimization*. CRC Press, Boca Raton, FL, 2010.

[25] H. Chou, M. Unger, and S.R. Quake. A microfabricated rotary pump. *Biomedical Microdevices*, 3, 2001.

[26] P. Coussy and A. Morawiec. *High-level synthesis: From algorithm to digital circuit*. Springer, 2008.

[27] P. S. Dittrich and A. Manz. Lab-on-a-chip: microfluidics in drug discovery. *Nature Reviews, Drug Discovery*, 5:210–218, 2006.

[28] J. El-Ali, P. K. Sorger, and K. F. Jensen. Cells on chips. *Nature*, 442:403–411, 2006.

[29] C. D. Chin et. al. Microfluidics-based diagnostics of infectious diseases in the developing world. *Nature Medicine*, 17:1015–1019, 2011.

[30] S. Einav et. al. Discovery of a hepatitis c target and its pharmacological inhibitors by microfluidic affinity analysis. *Nature Biotechnology*, 12:1019–1027, 2008.

[31] R. B. Fair. Digital microfluidics: is a true lab-on-a-chip possible? *Microfluidics and Nanofluidics*, 3(3):245–281, 2007.

[32] H. C. Fan, Y. J. Blumenfeld, U. Chitkara, L. Hudgins, and S. R. Quake. Noninvasive diagnosis of fetal aneuploidy by shotgun sequencing dna from maternal blood. *Proceedings of the National Academy of Sciences USA*, 105(42):16266–16271, 2008.

[33] C. Fang, Y. Wang, N. T. Vu, W. Lin, Y. Hsieh, L. Rubbi, M. E. Phelps, M. Mueschen, Y. Kim, A. F. Chatziioannou, H. Tseng, and T. G. Graeber. Integrated microfluidic and imaging platform for a kinase activity radioassay to analyze minute patient cancer samples. *Cancer Research*, 70(21):8299–8308, November 2010.

[34] F. Glover and M.Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[35] C. L. Hansen, M. O. A. Sommer, and S. R. Quake. Systematic investigation of protein phase behavior with a microfluidic formulator. *Proceedings of the National Academy of Sciences USA*, 101(40):14431–14436, 2004.

[36] D. J. Harrison, K. Fluri, K. Seiler, Z. Fan, C. S. Effenhauser, and A. Manz. Micromachining a miniaturized capillary electrophoresis-based chemical analysis system on a chip. *Science*, 261:895–897, 1993.

[37] A. Hertz and D. Werra. Using Tabu search techniques for graph coloring. *Journal of Computing*, 39:345–351, 1987.

[38] J. W. Hong, Y. Chen, W. F. Anderson, and S. R. Quake. Molecular biology on a microfluidic chip. *Journal of Physics: Condensed Matter*, 18(18):691–701, 2006.

[39] J. W. Hong and S. R. Quake. Integrated nanoliter systems. *Nature Biotechnology*, 21:1179–1183, 2003.

[40] J. W. Hong, V. Studer, G. Hang, W. F. Anderson, and S. R. Quake. A nanoliter-scale nucleic acid processor with parallel architecture. *Nature Biotechnology*, 22(4):435–439, 2004.

[41] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM (JACM)*, 45:246–265, 1998.

[42] M. E. Kempner and R. A. Felder. A review of cell culture automation. *The Journal of the Association for Laboratory Automation (JALA)*, 7(2):56–62, 2002.

[43] I. Klammer, A. Buchenauer, H. Fassbender, R. Schlierf, G. Dura, W. Mokwa, and U. Schnakenberg. Numerical analysis and characterization of bionic valves for microfluidic pdms-based systems. *Journal of Micromechanics and Microengineering*, 17(7):S122–S127, 2007.

[44] M. F. Kramer and D. M. Coen. Enzymatic amplification of dna by pcr: Standard procedures and optimization. *Current Protocols in Molecular Biology)*, pages 15.1.1–15.1.14, 2001.

[45] K. Kuchcinski. Constraints-driven scheduling and resource assignment. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 8(3):355–383, 2003.

[46] H. P. Le. Progress and trends in ink-jet printing technology. *Journal of Imaging Science and Technology*, 42:49–62, 1998.

[47] C. C. Lee, A. Elizarov, C. J. Shu, Y. S. Shin, and A. N. Dooley. Multistep synthesis of a radiolabeled imaging probe using integrated microfluidics. *Science*, 310:1793–1796, 2005.

[48] A. Lim, Y. Zhu, and Q. Lou. Heuristic methods for graph coloring problems. In *ACM symposium on Applied Computing*, pages 933–939, 2005.

[49] Y. C. Lim, A. Z. Kouzani, and W. Duan. Lab-on-a-chip: a component view. *Journal of microsystems technology*, 16(12), December 2010.

[50] H. T. G. Lintel. A piezoelectric micropump based on micromachining of silicon. *Sensors and Actuators*, 15(2):153–167, 1988.

[51] J. Liu, C. Hansen, and S. R. Quake. Solving the "world-to-chip" interface problem with a microfluidic matrix. *Analytical Chemistry*, 75(18):4718–4723, 2003.

[52] A. Manz, N. Graber, and H. M. Widmerl. Miniaturized total chemical analysis systems: A novel concept for chemical sensing. *Sensors and Actuators B: Chemical*, 1:244–248, 1990.

[53] J. S. Marcus, W. F. Anderson, and S. R. Quake. Microfluidic single-cell mrna isolation and analysis. *Analytical Chemistry*, 78(9):3084–3089, 2006.

[54] D. Mark, S. Haeberle, G. Roth, F. Stetten, and R. Zengerle. Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Chem. Soc. Rev.*, 39:1153–1182, 2010.

[55] J. Melin and S. Quake. Microfluidic large-scale integration: The evolution of design rules for biological automation. *Annual Reviews in Biophysics and Biomolecular Structure*, 36:213–231, 2007.

[56] G. D. Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, New York, 1994.

[57] W. H. Minhass, P. Pop, and J. Madsen. Application-specific architectural synthesis for the flow-based microfluidic large-scale integration biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (preparing for submission)*.

[58] W. H. Minhass, P. Pop, and J. Madsen. System-level modeling and application mapping on the flow-based microfluidic very large scale integration biochips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (preparing for submission)*.

[59] W. H. Minhass, P. Pop, and J. Madsen. System-level modeling and synthesis of flow-based microfluidic biochips. In *Proc. of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES)*, 2011.

[60] W. H. Minhass, P. Pop, and J. Madsen. Synthesis of Biochemical Applications on Flow-Based Microfluidic Biochips using Constraint Programming. In *Proc. of the IEEE Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*, pages 37–41, 2012.

[61] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga. Architectural synthesis of flow-based microfluidic large-scale integration biochips. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES)*, pages 181–190, 2012.

[62] W. H. Minhass, P. Pop, J. Madsen, M. Hemmingsen, and M. Dufva. System-level modeling and simulation of the cell culture microfluidic biochip ProCell. In *Proc. of the IEEE Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*, pages 91–98, 2010.

[63] W. H. Minhass, P. Pop, J. Madsen, M. Hemmingsen, P. Skafte-Pedersen, and M. Dufva. Cell Culture Microfluidic Biochips: Experimental Throughput Maximization. In *Proc. of the IEEE International Conference on Bioinformatics and Biomedical Engineering (iCBBE)*, 2011.

[64] W. H. Minhass, P. Pop, J. Madsen, and T. Ho. Control synthesis and pin count minimization for the flow-based microfluidic large-scale integration biochips. *Journal on Emerging Technologies in Computing Systems (preparing for submission)*.

[65] W. H. Minhass, P. Pop, J. Madsen, and T. Ho. Control Synthesis for the Flow-Based Microfluidic Large-Scale Integration Biochips. In *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013.

[66] J. M. Perkel. Microfluidics - bringing new things to life science. *Science*, November 2008.

[67] M. G. Pollack, A. D. Shenderov, and R. B. Fair. Electrowetting-based actuation of droplets for integrated microfluidics. *Lab Chip Journal*, 2:96–101, 2002.

[68] P. Pop, E. Maftei, and J. Madsen. Recent research and emerging challenges in the system-level design of digital microfluidic biochips. In *Proc. of the IEEE International SOC Conference (SOCC)*, 2011.

[69] S. M. Sait and H. Youssef. *VLSI physical design automation: theory and practice*. World Scientific Publishing Co. Pte. Ltd., 1999.

[70] M. F. Schmidt, W. H. Minhass, P. Pop, and J. Madsen. Modeling and simulation framework for flow-based microfluidic biochips. In *Proc. of the IEEE Symposium on Design, Test, Integration and Packaging of MEMS/MOEMS (DTIP)*, 2013.

[71] C. Schulte, G. Tack, and M. Z. Lagerkvist. Modeling and programming with gecode. `http://www.gecode.org/`.

[72] S. Shoji, M. Esashi, and T. Matsuo. Prototype miniature blood gas analyser fabricated on a silicon wafer. *Sensors and Actuators*, 14(2):101–107, 1988.

[73] J. Siegrist, M. Amasia, N. Singh, D. Banerjee, and M. Madou. Numerical modeling and experimental validation of uniform microchamber filling in centrifugal microfluidics. *Lab Chip*, 10:876–886, 2010.

[74] O. Sinnen. *Task Scheduling for Parallel Systems*. John Wiley & Sons Inc., Hoboken, New Jersey, 2007.

[75] G. Sittampalam, S. Kahl, and W. Janzen. High-throughput screening: advances in assay technologies. *Curr Opin Chem Biology*, 1(3):384–391, 1997.

[76] P. Skafte-Pedersen, M. Hemmingsen, D. Sabourin, F. S. Blaga, H. Bruus, and M. Dufva. A self-contained, programmable microfluidic cell culture system with real-time microscopy access. *Biomed Microdevices*, 14(2):385–399, 2012.

[77] A. Stacey and G. Stacey. *Routine quality control testing of cell cultures, Antiviral Methods and Protocols (3)*. Springer, 2000.

[78] F. Su and K. Chakrabarty. High-level synthesis of digital microfluidic biochips. *Journal on Emerging Technologies in Computing Systems*, 3(4), January 2008.

[79] F. Su, S. Ozev, and K. Chakrabarty. Concurrent testing of droplet-based microfluidic systems for multiplexed biomedical assays. In *Proceedings of the International Test Conference (ITC)*, pages 883–892, 2004.

[80] S. C. Terry, J. H. Jerman, and J. B. Angell. A gas chromatographic air analyzer fabricated on a silicon wafer. *IEEE Transactions on Electron Devices*, 26(12):1880–1886, 1979.

[81] W. B. Thies. Programmable microfluidics. *presented at Stanford University*, October 2007.

[82] T. Thorsen, S. J. Maerki, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, October 2002.

[83] D. Ullman. Np-complete scheduling problems. *Journal of Computing System Science*, 10:384–393, 1975.

[84] D. Ullman. NP-complete scheduling problems. *Journal of Computing System Science*, 10:384–393, 1975.

[85] J. P. Urbanski, W. Thies, C. Rhodes, S. Amarasinghe, and T. Thorsen. Digital microfluidics using soft lithography. *Lab Chip*, 6:96–104, 2006.

[86] V. T. Vanchikov. Special form of laminar liquid flow in hydraulic devices. *Russian Engineering Research*, 28(9):854–855, 2008.

[87] G. M. Whitesides. The origins and the future of microfluidics. *Nature*, 442:368–373, July 2006.

[88] Y. Zhao and K. Chakrabarty. Cross-contamination avoidance for droplet routing in digital microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31:817–830, June 2012.